Titre: Title:	Analyse et détermination de codes doublement orthogonaux pour décodage itératif
Auteur: Author:	Brice Baechler
Date:	2000
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Baechler, B. (2000). Analyse et détermination de codes doublement orthogonaux pour décodage itératif [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8602/

UNIVERSITÉ DE MONTRÉAL

ANALYSE ET DÉTERMINATION DE CODES DOUBLEMENT ORTHOGONAUX POUR DÉCODAGE ITÉRATIF

BRICE BAECHLER
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE ET DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES (M.Sc.A.)
(GÉNIE ÉLECTRIQUE)
JUIN 2000

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

ANALYSE ET DÉTERMINATION DE CODES DOUBLEMENT ORTHOGONAUX POUR DÉCODAGE ITÉRATIF

présenté par : BAECHLER Brice

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

Mme. JAUMARD Brigitte, Ph.D., Thèse d'habilitation, présidente

M. HACCOUN David, Ph.D., membre et directeur de recherche

M. GAGNON François, Ph.D., membre et codirecteur de recherche

Mme. BELLAÏCHE Martine, M.Sc.A., membre

À Caroline et Christian mes parents.

REMERCIEMENTS

Le travail de recherche présenté dans ce mémoire n'aurait pu être mené à bien sans l'aide et le soutien de nombreuses personnes que je tiens à remercier à travers ces quelques lignes.

Tout d'abord je tiens à exprimer ma gratitude envers mon directeur de recherche Dr. David Haccoun et mon co-directeur de recherche Dr. François Gagnon. Le temps qu'ils ont su m'accorder ainsi que les différents conseils et directives qu'ils m'ont prodigués ont été d'une grande aide et m'ont permis d'aboutir aux résultats présentés. Je les remercie également de m'avoir accorder une aide financière pendant la majeure partie de ma recherche.

J'aimerais également exprimer ma reconnaissance aux personnes de l'École Supérieure d'Électricité (Supélec) en France qui m'ont assisté tout au long de cette formation "double diplômante". Je pense naturellement à Mme Dao-Duy et au Dr. Gilles Fleury, mon coordinateur.

Mes remerciements s'adressent également à tous mes collègues de travail qui ont su créer une ambiance dynamique au sein du laboratoire. Merci à Steve, Youenn, Pierre et Khaled. Je remercie plus particulièrement Lionel Scremin, Pierre-Fréderic Caillaud, Guillaume Boillet et Christian Cardinal pour les discussions fructueuses que nous avons eues.

Je tiens finalement à exprimer ma gratitude envers mes proches et mes amis pour leur soutien pendant mon séjour à Montréal. Je pense en particulier à mes parents Caroline et Christian, à mon amie Stéphanie et à mes amis.

RÉSUMÉ

Le travail de recherche présenté dans ce mémoire concerne la génération d'une nouvelle classe de codes dénommés codes doublement orthogonaux. Ceux ci utilisés pour fiabiliser les transmissions de données ont été introduits il y a quelques années. Ils sont en fait la pièce maîtresse d'un nouveau système de codage et de décodage qui représente une évolution importante du principe turbo traditionnel. Pour des performances voisines à celles du plus efficace des codeurs et décodeurs turbo actuels, ce système ne requiert qu'une bien moindre implémentation matérielle réduisant ainsi son coût de fabrication. Cette amélioration obtenue au prix de la suppression d'un élément essentiel de la structure du codage turbo (l'entrelaceur) reporte toute la complexité du système dans les spécificités des codes à utiliser. La double orthogonalité qui est la condition à satisfaire par ces codes peut être vue mathématiquement comme une extension du problème des règles de Golomb à un ordre supérieur. Ainsi, le champ d'applications de certaines parties de cette étude va au delà du seul domaine des télécommunications.

Notre but a donc été de générer de tels codes doublement orthogonaux de la manière la plus rapide et la plus efficace possible. La longueur du code ("span") qui détermine la latence du système et le niveau de mémoire requis est un facteur des plus influents. On se doute en effet de l'importance de ces deux paramètres dans le cadre des communications mobiles. Nous nous sommes ainsi attachés à obtenir pour cette grandeur une valeur la plus réduite possible.

La première partie de l'étude qui fut essentiellement théorique nous a permis, en nous basant sur les travaux antérieurs de bien définir les caractéristiques mathématiques que devait vérifier cette classe de codes. A partir de là des propriétés remarquables de ces codes ont été établies et démontrées ainsi que cer-

taines définitions complétées. Cette analyse a permis de différencier deux sortes de codes doublement orthogonaux, ceux au sens strict et ceux au sens large. Leurs caractéristiques différentes nous ont conduit à effectuer deux études distinctes tout en observant un cheminement similaire.

La phase de génération qui est celle qui fut abordée en premier nous donna l'occasion de revenir sur les méthodes précédemment employées dans le but de les améliorer et également d'en présenter de nouvelles basées pour les plus intéressantes sur l'application d'un critère aléatoire. Ce type de méthodes originales se démarque des solutions précédemment envisagées qui faisaient appel à de solides théories algébriques. Les différentes valeurs générées furent une première base de comparaisons et de tests. Pour compléter cette étape, une phase de réduction a ensuite été implémentée pour tenter de minimiser la longueur des codes obtenus. À la suite de l'analyse théorique de différentes stratégies, de nouvelles solutions algorithmiques ont été adoptées et implémentées. La conjugaison des deux phases a conduit à des résultats intéressants.

Les comparaisons effectuées avec pour base les méthodes de génération et de réduction précédemment connues font apparaître des gains importants au niveau de la vitesse d'exécution, de la simplicité d'implémentation et de l'efficacité. Les longueurs maximales des codes générés ont été considérablement réduites avec un facteur de gain supérieur à 60 dans certains cas. Il a même été possible de s'attaquer à des codes doublement orthogonaux au sens strict de taux relativement élevé ($\geq \frac{6}{12}$), codes qui n'avaient pu être générés par le passé.

Dans chaque cas, des simulations ont été réalisées pour vérifier la validité des codes générés et le niveau de performance atteint. Celles-ci nous ont permis de tirer des conclusions importantes et des faits remarquables quant à la distinction des

deux types différents de codes présentés. Des travaux complémentaires au niveau de la longueur minimale théoriquement atteignable et du temps de réduction mis en jeu ont également été abordés.

Les bons résultats obtenus ne doivent cependant pas faire oublier qu'une théorie satisfaisante n'a pu être établie de manière exacte et que rien ne nous permet à ce jour d'être sûr d'atteindre des codes aux longueurs minimales. Cependant les améliorations importantes introduites permettent d'élargir le champ d'applications du nouveau système de codage. En réduisant de manière importante les longueurs des codes utilisés, l'on diminue dans les mêmes proportions la latence du système et le niveau de mémoire nécessaire.

ABSTRACT

This study concerns the search and determination of a new classe of codes called Convolutional Self Doubly Orthogonal Codes. These codes may be advantageously utilised for a novel coding/decoding technique recently introduced as an amelioration of Turbo Codes encoding.

Turbo codes present the advantage of remarkable error performances. Since their introduction in 1993, they have been the object of intense theoretical and practical investigations. Nevertheless the material structure remains complex with the use of at least two encoders/decoders and two interleavers. The presence of the interleavers and the high number of iterations required to achieve very good performances produce an inherent latency unsuitable for some "real-time" applications. Moreover the decoding procedure suffers from a substantial complexity.

The amelioration which circumvents both the decoding complexity and interleaver requirements of the usual Turbo Codes is based on new orthogonal threshold decodable convolutional codes and a modified threshold decoder which is used iteratively. It employs a single encoder only, a single decoder and no interleaver. Such an approach requires using threshold decodable codes which must exhibit further orthogonal properties than the well known orthogonal codes discovered in 1963. Our work has been to determinate and generate these codes.

The first stage was a theorical study. By analysing publications on this subject, we determined the definition and the characteristics of these codes. We made the distinction between the orthogonality in the wide and in the strict sense. We were then able to isolate and demonstrate some remarkable properties that were used to implement several methods of generation. We introduce some original concepts of

generation by using pseudo random constructive methods.

Since the code constraint length correspond to the latency of each decoding iteration, an important parameter in the code searching, is the minimisation of the code constraint length for a given error correcting capability. We were then interested in finding the code with the minimal length for a number of generators given. Once the generation is accomplished, improvement on the code length is attempted by using reduction methods that exploit the specifities of double orthogonality.

To complete this work, we analyzed the computation time required and the limitations of our algorihms. The error performances of our codes have been simulated using the novel iterative decoding algorithm.

The code generation method and its ensuing reduction procedure has yielded good novel convolutional self doubly orthogonal codes without requiring an excessive computation time. The length difference between the codes previously generated and the ones we obtained demonstrates clearly the substantial advantages of our novel generation/reduction procedures.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET DES SYMBOLES

C. S. O^2. C. :	convolutional self doubly orthogonal codes
d.o. :	doublement orthogonal
T.D.M.A :	time division multiple access
C.D.M.A :	code division multiple access
MAP :	Maximum A Postériori
long. :	longueur
nb. :	nombre
gen. :	générateur
tmps. :	temps
opt. :	optimisé(s)
Pb. :	probabilité
min. :	minimum
h :	heure(s)
' :	minute(s)
" :	seconde(s)

LISTE DES ANNEXES

CHAPITRE 1

INTRODUCTION

L'atmosphère dans laquelle nous évoluons est balayée par des ondes, les routes que nous empruntons sont irriguées de câbles souterrains en cuivre ou de fibres optiques. Le réseau de télécommunications est omniprésent, mais demeure le plus souvent invisible. Pourtant cette technologie influence notre façon de travailler, de nous divertir, bref de vivre. Qui aurait pu croire lorsque M. Bell fit la démonstration de son téléphone que les télécommunications connaîtraient un tel essor. L'évolution rapide de ces dernières années est à mettre principalement sur le compte de l'introduction de l'informatique. La puissance et la rapidité de calcul ont en effet ouvert la voie à de nouvelles applications et ont permis la gestion de systèmes de plus en plus importants. Parmi les développements les plus remarquables, on notera le réseau Internet qui était encore quasiment méconnu il y a cinq ans à peine et dont le trafic double à présent tous les trois mois et la téléphonie cellulaire dont le taux de pénétration ne cesse de croître. Difficile, voire impossible, dans ces conditions de prévoir les développements futurs. On peut juste signaler à plus ou moins court terme la fusion attendue des services Internet aux unités mobiles et la généralisation des réseaux câblés à large bande. Les communications par satellite pour systèmes mobiles n'ont quant à elles pour l'instant pas le succès escompté. Cette technologie se heurte à de nombreux problèmes avec particulièrement d'énormes coûts de mise en œuvre.

Le développement des télécommunications ne se fait pas sans difficultés et plus que tout, aujourd'hui, ce sont les problèmes de réalisations matérielles qui freinent l'expansion de ces technologies. Tous les outils pour réaliser des systèmes de communication fiables, rapides et accessibles à tous ne sont pas encore tout à fait maîtrisés.

Le domaine du codage est un point essentiel d'une chaîne de transmission. En codant l'information avant sa transmission cette technique permet de détecter et dans certains cas de corriger les erreurs ou de remédier aux pertes survenues lors du transport des données. C'est en ce sens un gage de fiabilité de la transmission. En contrepartie le temps et le matériel ainsi que la largeur de bande mis en jeu pour parvenir à un bon niveau de performances peut s'avérer préjudiciable dans le cadre de nombreuses applications. Ce dernier point justifie les diverses recherches entreprises dans ce domaine.

Le système de codage le plus performant à l'heure actuelle est dénommé "codage turbo". Il n'a cessé depuis son introduction en 1993 d'évoluer et de s'améliorer. Cependant la qualité des performances atteintes est obtenue au prix d'une certaine complexité matérielle et d'un délai important généré lors de l'étape de décodage. Ces deux faits sont en réalité deux limitations importantes restreignant le champ d'applications de ce système. à celles-ci s'ajoute un coût de réalisation relativement élevé dû à une complexité matérielle plus importante que celle d'autres systèmes. Cependant, le niveau de performances atteint est tel que malgré ses inconvénients, ce type de codage fait partie de la norme des prochains systèmes cellulaires CDMA à large bande.

Une évolution différente des améliorations qui avaient été jusque là envisagées pour ce système fut proposée par Messieurs François Gagnon et David Haccoun en 1997. Leur approche du problème était relativement radicale dans le sens où en supprimant une partie matérielle essentielle du principe turbo (l'entrelaceur) tout en conservant un niveau de performances voisin, ils réduisaient de façon importante la complexité et le délai du système. On entrevoit immédiatement les avantages que l'on pourrait tirer d'une telle amélioration qui arrive peu à peu à maturité.

L'idée introduite pour obtenir ce résultat consistait à reporter la complexité matérielle au niveau des codes utilisés. Ces derniers doivent donc disposer de propriétés relativement complexes pour espérer compenser les modifications matérielles adoptées. Après avoir défini et établi sous forme théorique le type de codes recherchés (dénommés "codes doublement orthogonaux"), les premières simulations furent implémentées. Les résultats obtenus apparurent très encourageants. Il subsistait cependant certaines limitations.

Un des principaux défauts que l'on pouvait reprocher à ce système était l'utilisation d'un ensemble de générateurs de longueur de contrainte élevée. Cette dernière grandeur qui conditionne la latence et le niveau de mémoire nécessaire au décodage est d'une grande importance et il y a tout intérêt à l'optimiser pour minimiser les besoins matériels et le délai d'attente de transmission.

Le cadre de notre étude s'inscrit dans la complémentarité des travaux précédemment réalisés sur ce nouveau type de codage. Plus particulièrement, nous nous sommes concentrés sur la détermination des codes nécessaires au bon fonctionnement de ce système. Nous avons essayé d'obtenir ceux disposant d'une longueur minimale en mettant en oeuvre diverses méthodes de génération et de réduction.

Ce mémoire est structuré comme suit :

Le chapitre 2 comprend une description de la discipline du codage. Un bref historique y est présenté avant de s'attarder sur les deux grands types de codage. Le cas des codes turbo est ensuite exposé plus en détail en termes de fonctionnement et de propriétés.

Le chapitre 3 décrit l'évolution importante introduite par M. Gagnon et M. Haccoun. Le concept de double orthogonalité est présenté au cours de ce chapitre avec un bref retour sur celui de simple orthogonalité. Certaines caractéristiques et propriétés remarquables des codes recherchés y sont présentées et démontrées.

Le chapitre 4 regroupe les différents travaux qui ont été menés à bien pour étudier le cas des codes doublement orthogonaux au sens large de taux $\frac{1}{2}$. Il se divise en plusieurs parties avec dans un premier temps l'exposé de méthodes de génération. Une étude de la longueur minimale est ensuite présentée avant d'introduire les différentes techniques de réduction utilisées. Les résultats ainsi que les simulations réalisées sont commentés avant de conclure.

Le chapitre 5 reprend en partie le cheminement choisi au chapitre 4 pour étudier le cas des codes doublement orthogonaux de taux $\frac{J}{2J}$ au sens strict. Seule l'analyse de la longueur minimale n'est pas reprise et une brève définition de ce type de code et de ses caractéristiques y est ajoutée.

Le chapitre 6 concerne l'étude des codes en blocs doublement orthogonaux. Là encore, il sera présenté l'étape de génération et de réduction après avoir établi certaines équivalences et certaines propriétés de cette classe de codes.

Le chapitre 7 conclut l'étude réalisée. Il contient également les points qu'il serait intéressant de poursuivre et d'approfondir dans le cadre d'un travail futur.

Les contributions apportées par ce travail de recherche sont les suivantes :

1. Apports théoriques aux niveaux des propriétés des codes doublement ortho-

gonaux :

- simplification de la condition de double orthogonalité,
- équivalence différences - sommes,
- dénombrement des différentes différences mises en jeu.

2. Élaboration théorique et pratique de méthodes de génération pour les trois types de codes considérés (codes doublement orthogonaux au sens large de taux $\frac{1}{2}$, au sens strict de taux $\frac{J}{2J}$ et en blocs). Rappels sur celles précédemment utilisées et descriptions des nouvelles retenues.

3. Conception théorique et pratique de méthodes de réduction pour chaque type de codes. Étude temporelle et algorithmique des solutions adoptées.

4. Analyse et obtention de bornes concernant la longueur minimale dans le cas des codes doublement orthogonaux au sens large de taux $\frac{1}{2}$.

5. Génération et présentation de nouveaux codes doublement orthogonaux aux longueurs réduites pour les trois types de codes considérés.

6. Analyse de performances et caractéristiques des codes obtenus.

7. Comparaisons des deux types d'orthogonalité (au sens large et au sens strict).

D'un point de vue pratique, un ordinateur de type *pentium*© *II* cadencé à une vitesse de 300Mhz et possédant 64Mb de RAM a été quasi-exclusivement utilisé. Quelques calculs intermédiaires ont été réalisé sur des stations SUN SPARC 4 et 10 et un autre PC cadencé lui à 350Mhz. Les logiciels de programmation *Visual*© *C++*

et *Borland*© *C* + + ainsi que le logiciel de calcul formel *Maple*© ont servi à l'implémentation des différentes procédures sous environnement Windows, g++ a également été utilisé sous environnement Linux.

CHAPITRE 2

CODAGE

2.1 Généralités

2.1.1 Définition

On distingue deux types de codage ayant des fonctions différentes : le codage de source et le codage de canal. Le codage de source sert généralement à représenter le signal d'une source sous une forme la plus efficace possible c'est à dire en éliminant les redondances. Ce rapport se préoccupe plutôt du codage de canal qui lui sert à détecter et si possible à corriger les erreurs qui peuvent survenir lors de la transmission de données.

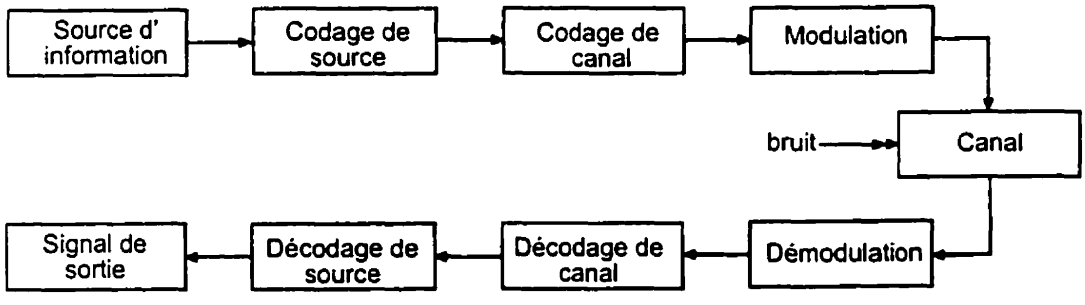

Figure 2.1 – *Chaîne de transmission générale*

L'étape de codage de canal consiste en général à affecter des symboles ou des séquences de symboles à chacun des messages délivrés par la source. Ces séquences contiennent un nombre d'éléments supérieur à celui a priori nécessaire pour exprimer de manière fidèle l'information initiale. Cette redondance ajoutée volontairement peut alors être astucieusement exploitée pour améliorer la fiabilité des

échanges. Cette qualité est cependant obtenue au détriment de la largeur de bande utilisée (le nombre de signaux pouvant être transmis par le canal est plus grand que le nombre de messages). On répertorie différentes méthodes d'affectation dont deux principales : le codage en blocs et le codage convolutionnel.

Pour une grande majorité de canaux, le théorème du codage de canal bruité (discipline de la théorie de l'information) stipule que si la capacité du canal (bit par symboles codés) est plus grande que l'entropie de la source, alors la probabilité d'erreur au décodage peut être rendue aussi proche de zéro qu'on le souhaite en augmentant la complexité du code. Malheureusement la théorie n'explicite en aucune façon la manière d'y parvenir. À ce jour, bien que l'on soit fort près (à env. 0.2 dB) en pratique de la borne théorique établie par Shannon [14], celle-ci n'a toujours pas été atteinte.

2.1.2 Historique

Le codage est une discipline qui remonte à la fin des années 1940. Shannon [14] dont le travail et les premières publications furent à l'origine de nombreuses considérations sur le canal de transmission peut être considéré comme le père de la théorie de l'information. Son approche du problème de communication à travers un canal bruité sur un plan statistique a eu un énorme impact et a généré de nombreux travaux de recherche. Hamming [15], collègue de Shannon au sein des laboratoires de la compagnie de téléphone Bell, adopta lui une vision plus combinatoire et fut, en ce sens, l'instigateur de la théorie du codage. Il fut le premier à formaliser un code qui s'avérait en mesure de corriger une unique erreur. Cette étude fut le point de départ de nombreuses recherches avec l'apparition de concepts tels que : le décodage en blocs, les bornes sur les distances, le poids d'un code... Golay [16] forme avec Hamming et Shannon un trio de pionniers dans le domaine du codage.

De nombreux travaux furent produits dans les années suivantes notamment au niveau de l'analyse des performances. Les années 1960 à 1970 virent apparaître de multiples contributions principalement focalisées sur les algorithmes de codage et de décodage.

Une découverte importante fut effectuée par Elias [17] en 1955 qui introduisit le concept des codes convolutionnels; codes qui s'avéraient être autant voire plus performants que leurs homologues en blocs pour une complexité d'implémentation bien moindre. Seul le décodage posait de gros problèmes. On note parmi les travaux effectués à ce niveau ceux de J. M. Wozencraft [2] avec l'introduction du décodage séquentiel et ceux de R. M. Fano [18] qui ont complété cette technique. Il a été montré que la probabilité d'erreur de tels décodeurs peut devenir voisine de zéro très rapidement.

La discipline du codage bénéficia grandement des recherches entreprises pour l'exploration spatiale dans les années qui suivirent. Le besoin de communications fiables et rapides était en effet mis en avant. L'algorithme de Viterbi [19] présenté en 1967 et les contributions apportées par Forney [20] ont ouvert la voie à des réductions substantielles au niveau de la longueur des décodeurs utilisés.

Le codage turbo [22] introduit en 1993 et qui sera détaillé plus précisément à la fin de ce chapitre représente ce qui se fait de mieux actuellement au niveau des performances d'erreur atteintes.

2.1.3 Utilisations

Le champ d'applications de la technique du codage est extrêmement vaste. Elle fait partie intégrante de la chaîne de transmission précédemment présentée. Tous les nouveaux systèmes échangeant des données utilisent ce principe pour garantir l'intégrité de leurs communications. On peut citer pour exemple des domaines d'applications tels que les réseaux cellulaires mobiles, les transmissions par satellite, les échanges d'information entre périphériques d'un ordinateur, les communications spatiales...

2.2 Codes en blocs et codes convolutionnels

Dans cette section, les deux grandes classes de codes sont décrites en termes de notations, de propriétés et de fonctionnement. Seules les notions nécessaires pour la suite de ce mémoire sont abordées. Le lecteur est prié de consulter les excellentes œuvres de référence traitant du codage dont [27].

2.2.1 Codes en blocs linéaires

2.2.1.1 Introduction

Les codes en blocs furent le premier type de code étudié. Ils concentrèrent l'attention de très nombreuses personnes pour des applications diverses. Leur faible complexité relative et leur structure mathématique ont grandement aidé leur développement. Si la majorité des codes particuliers basés sur le codage en blocs avait pour but de corriger les erreurs de transmission ou de stockage, certains ont été également conçus pour corriger les erreurs de synchronisation [23] ou même, dans un cadre plus général, les erreurs arithmétiques.

2.2.1.2 Présentation

Un code en blocs consiste en un ensemble de vecteurs de longueur donnée, appelés "mots de code". Les éléments constitutifs des vecteurs sont issus d'un certain alphabet Q (de cardinalité q) ($\{0,1\}$ pour un alphabet binaire) et leur nombre n au sein d'un vecteur est par définition la longueur du code. Il y a donc q^n mots de code possibles avec les notations choisies. La séquence d'entrée est constituée de k symboles issus d'un alphabet P (de cardinalité p) ce qui génère p^k possibilités ($p^k < q^n$). Il est ainsi possible de sélectionner p^k mots de code parmi nos q^n pour les "associer" aux p^k séquences d'entrée. Les k symboles d'information sont donc "traduits" par un mot de code de longueur n symboles codés. Dans la majorité des cas les alphabets P et Q sont identiques et l'on a simplement $k < n$. La différence de longueur entre les séquences d'entrée et celles de sortie assure la présence de symboles supplémentaires utilisés à des fins de vérifications. Le taux de codage (noté R) de ce code en blocs caractérisé par le couplet (k,n) est alors défini par $\frac{k}{n}$. On peut formaliser cette définition par le schéma suivant :

$$\begin{cases} \textit{mot en entrée} \\ \textit{de longueur } k \\ \textit{issu de l'alphabet } p \end{cases} \Rightarrow \begin{cases} \textit{mot en sortie} \\ \textit{de longueur } n \\ \textit{issu de l'alphabet } q \end{cases}$$

On nomme "poids" d'un mot de code le nombre d'éléments différents de 0 que contient celui ci. L'ensemble de tous les poids constitue la distribution de poids du code. La distance entre deux mots de code est le nombre d'éléments différenciant chacun des deux mots. La plus petite de ces grandeurs est appelée distance minimale (d_{min}).

Les codes dits linéaires permettent d'utiliser des opérations d'additions et de multiplications sur les mots de code au niveau des étapes de codage et de décodage.

Ces opérations sont effectuées selon une certaine arithmétique. Cette dernière est définie suivant les lois régissant le corps fini dont les éléments constituent l'alphabet Q. On reviendra au chapitre 4 (4.2.1.1) sur ces corps au nombre d'éléments fini connus sous le nom de corps de Galois. L'application de la théorie qui leur est associée aux problèmes des communications [27] date de la fin des années 1950.

2.2.1.3 Principe

Deux concepts importants pour bien appréhender la notion de codes en blocs sont la matrice génératrice ($G = (g_{i,j})$) et la matrice de parité (aussi dénommée matrice de contrôle) ($H = (h_{i,j})$). Soit $I_m = [i_{m,0}\ i_{m,1}\ \ldots\ i_{m,k-1}]$ le vecteur représentant les k symboles d'information considérés en entrée et $C_m = [c_{m,0}\ c_{m,1}\ \ldots\ c_{m,n-1}]$ le mot de code correspondant en sortie. L'opération d'encodage peut être représentée par un ensemble de n équations de la forme:

$$\begin{gathered} c_{m,j} = \sum_{l=0}^{k-1} g_{l,j}.i_{m,l} \\ c_{m,j} = g_{0,j}.i_{m,0} + g_{1,j}.i_{m,1} + \ldots + g_{k-1,j}.i_{m,k-1} \\ où\ g_{i,j} = 0\ ou\ 1\ avec\ j = 0,1,\ldots,n-1. \end{gathered} \tag{2.1}$$

Il est encore plus simple d'adopter une représentation matricielle du type:

$$C_m = I_m.G \tag{2.2}$$

où G la matrice génératrice est de la forme :

$$G = \begin{bmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,n-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,n-1} \\ \cdots & \cdots & \cdots & \cdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{bmatrix} \tag{2.3}$$

La matrice G doit naturellement être de rang k. Celle ci peut être réduite par de simples opérations (additions et soustractions) sur les lignes et des permutations sur les colonnes à une matrice sous la forme dite systématique :

$$G = [I_k|P] = \left[\begin{array}{ccccc|ccccc} 1 & 0 & 0 & \ldots & 0 & p_{0,0} & p_{0,1} & \cdots & p_{0,n-k-1} \\ 0 & 1 & 0 & \ldots & 0 & p_{1,0} & p_{1,1} & \cdots & p_{1,n-k-1} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & 1 & p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} \end{array}\right] \tag{2.4}$$

La matrice P met en évidence la redondance ajoutée pour augmenter la fiabilité (on a ainsi pour $j \geq k$, $c_{m,j} = \sum_{l=0}^{k-1} p_{l,j-k}.i_{m,l}$). On définit la matrice H, matrice de parité telle que : $G.H' = 0$ (H' transposée de H). En adoptant la notation "systématique", on obtient $H = [-P'|I_{n-k}]$. Cette matrice est importante surtout au niveau du décodage puisqu'elle permet de comparer les bits redondants reçus avec le résultat obtenu par combinaisons linéaires des bits systématiques reçus.

Illustrons ce dernier point par un exemple :

Soit un code (7,4) (notation explicitée à la page 11) avec une matrice génératrice définie comme suit :

$$G = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right] \tag{2.5}$$

En utilisant le fait que G est sous forme systématique, l'expression d'un mot de code sera : $C_m = [c_{m,0}\ c_{m,1}\ c_{m,2}\ c_{m,3}\ c_{m,4}\ c_{m,5}\ c_{m,6}] = [i_{m,0}\ i_{m,1}\ i_{m,2}\ i_{m,3}\ d_{m,0}\ d_{m,1}\ d_{m,2}]$. Les $d_{m,j}$ représentent les symboles de parité.

$$\begin{cases} d_{m,0} = i_{m,0} + i_{m,1} + i_{m,2} \\ d_{m,1} = i_{m,1} + i_{m,2} + i_{m,3} \\ d_{m,2} = i_{m,0} + i_{m,1} + i_{m,3} \end{cases} \tag{2.6}$$

Notons $y_{m,j}$ les symboles reçus. La matrice de parité H est décrite par :

$$H = \left[\begin{array}{cccc|ccc} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \tag{2.7}$$

Le produit $C_m.H'$ donne les équations de syndromes :

$$\begin{cases} i_{m,0} + i_{m,1} + i_{m,2} + d_{m,0} = 0 \\ i_{m,1} + i_{m,2} + i_{m,3} + d_{m,1} = 0 \\ i_{m,0} + i_{m,1} + i_{m,3} + d_{m,2} = 0 \end{cases} \tag{2.8}$$

Ainsi l'on peut s'assurer qu'un code reçu Y vérifie les conditions (2.6) en regardant si $Y.H' = 0$ (annulation des équations de syndromes).

On cite ci-après quelques exemples parmi les plus courants de codes en blocs

linéaires :

- Codes de Hamming binaires : $(n,k)=(2^m - 1,2^m - 1 - m)$; $(d_{min} = 3)$.
- Codes de Golay (23,12) ; $(d_{min} = 7)$.
- Codes BCH binaires (Bose-Chaudhuri-Hocquenghem) $n = 2^m - 1$, $n - k \leq mt$ et $d_{min} = 2t + 1$ (où m $(m \geq 3)$ et t sont deux entiers positifs arbitraires).
- Codes de Reed-Solomon (avec un alphabet de q symboles on a : $n = q - 1$, $k = 1,2,...,n - 1$, $d_{min} = n - k + 1$ et $R_c = \frac{k}{n}$). Un tel code est assuré de corriger $\lfloor \frac{d_{min}-1}{2} \rfloor$ erreurs.

2.2.2 Codes convolutionnels

2.2.2.1 Introduction

Les codes convolutionnels initialement introduits par P. Elias en 1955 [17] possèdent de nombreux avantages sur leurs homologues à blocs. Ils s'avèrent cependant beaucoup plus difficiles à analyser. L'introduction du décodage séquentiel [2] en 1961 fut un grand apport à cette technique. Les nombreux travaux effectués en particulier par G. D. Forney [20], [21] et R. M. Fano [18] ont accéléré leur développement et ils sont à ce jour massivement utilisés pour obtenir des systèmes pratiques et performants.

2.2.2.2 Fonctionnement

Les codes convolutionnels constituent une famille de codes correcteurs qui, à l'image des codes en blocs, génèrent pour chaque bit d'information un ensemble de symboles codés communément appelés symboles de parité. Ils reposent, contrairement aux codes en blocs, sur un codage continu de l'information à transmettre.

Ce type de codage se présente matériellement sous la forme d'un ou plusieurs registres à décalage dont une partie du contenu est reliée à des additionneurs modulo 2. Le registre se compose de K (v bits) niveaux et de z sorties. La grandeur v représente le nombre de bits d'information qui pénètrent dans le système à chaque décalage (coup d'horloge). On définit les générateurs notés g_i où i varie de 0 au nombre de sorties du système moins 1 et où g_i est constitué d'une succession de "0" et de "1" représentatif de l'absence ("0") ou de la présence ("1") d'une connexion entre les différents emplacements du registre et l'additionneur relié à la sortie i. On utilise usuellement au lieu de cette notation binaire l'équivalent sous forme octale (base numérique 8).

La figure 2.2 illustre ces définitions:

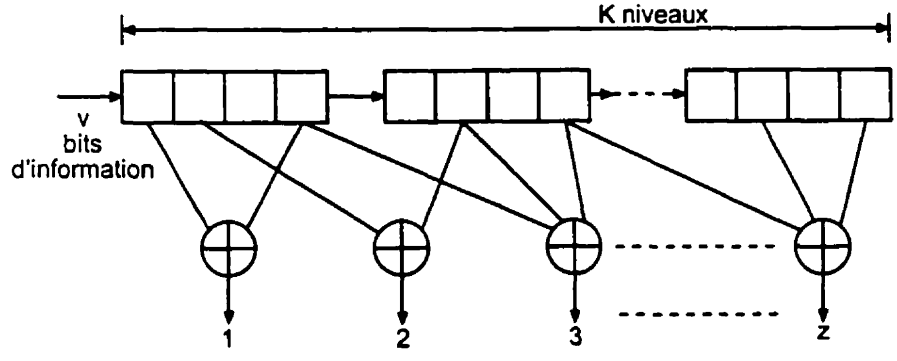

Figure 2.2 – *Illustration des notations employées*

On nomme longueur de contrainte la grandeur K. On nomme mémoire ou latence du codeur la valeur $\mu = K - v$. Cette grandeur est importante car plus elle est élevée et plus le temps d'attente demandé avant de traiter une séquence sera grand. En contrepartie, la performance d'erreur du système augmente de manière exponentielle avec μ.

Le taux de codage R est défini par $R = \frac{v}{z}$.

On note d_l le nombre d'éléments qui diffèrent entre deux "mots" de code considérés sur l symboles et qui possèdent deux symboles initiaux différents. On appelle $d_{min,l}$ la *lième* distance minimale définie par $d_{min,l} = mind_l$ pris sur l'ensemble du code. On nomme d_{libre} la distance libre de ce code définie par: $d_{libre} = \max_l\ d_l$.

Une façon simple de représenter un encodeur convolutionnel consiste en un schéma du type de ceux présentés dans les figures 2.3 et 2.4.

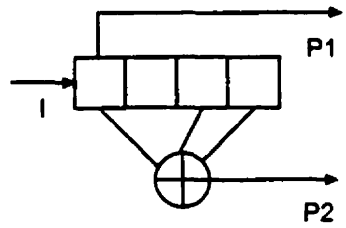

Figure 2.3 - *Codeur de taux* $\frac{1}{2}$ *(*$v = 1$, $z = 2$*), longueur de contrainte:* $K = 4$, $\mu = 3$, *générateur: 1011 (13 en octal)*

On dénote ce code comme étant systématique de par le fait qu'une des séquences de sortie (P1) reproduit exactement les symboles présents à l'entrée (I). L'analogie avec le cas des codes en blocs est évidente. On appelle "états" de l'encodeur les différentes combinaisons possibles du contenu du registre sans considérer la(es) case(s) d'entrée.

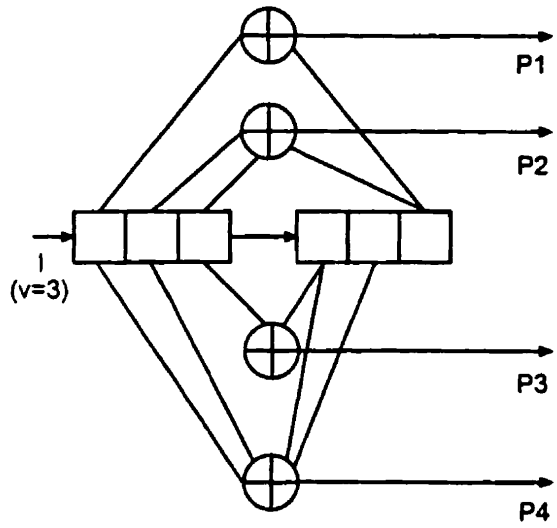

Figure 2.4 - *Codeur de taux* $\frac{3}{4}$ *(*$v = 3$, $z = 4$*), longueur de contrainte:* $K = 6$, $\mu = 3$, *générateurs: 110001 (61), 001100 (14) et 110110 (66)*

Pour étudier le comportement d'un code convolutionnel on peut utiliser trois méthodes distinctes:

- l'arbre,
- le treillis,
- le diagramme d'états.

On rappelle ci-après le fonctionnement de ces structures dans le cas de codes binaires (on utilisera l'exemple du codeur de taux $\frac{1}{2}$, $K = 4$ présenté à la figure 2.3):

- L'arbre

  Celui-ci se construit très facilement comme cela est illustré à la figure 2.5. Chaque branche "ascendante" correspond à l'évolution du système suite à l'entrée d'un "0" dans le codeur, chaque branche "descendante" correspond-elle à l'évolution du système suite à l'entrée d'un "1" dans le codeur.

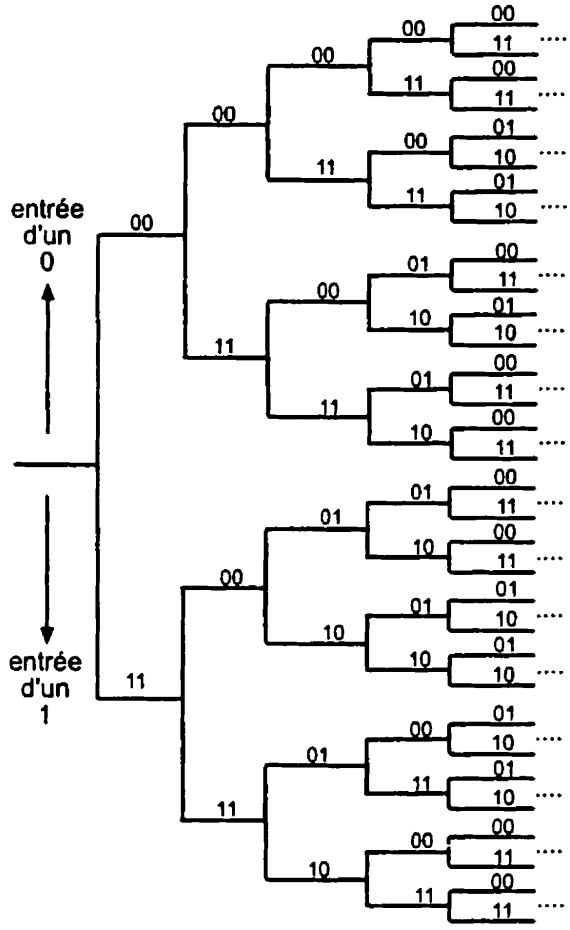

Figure 2.5 – *Arbre correspondant au codeur de taux* $\frac{1}{2}$, $K = 4$, *présenté à la figure 2.3*

Les chiffres inscrits sur chaque branche représentent la sortie de l'encodeur obtenue pour le bit d'entrée ayant conduit à la branche considérée. La structure de l'arbre se répète à partir du quatrième niveau (influence de la longueur de contrainte (ici 4)). En effet les deux symboles en sortie du codeur (2.3) à chaque étape sont déterminés par le bit d'entrée et les trois derniers bits présents dans le registre (le quatrième bit présent dans le registre n'a aucune influence puisqu'il "sort" dès qu'un nouveau bit entre). On définit ainsi pour notre exemple huit états différents dans lesquels peut se trouver le codeur (correspondant aux différentes combinaisons que peuvent prendre les trois

symboles binaires précédant l'entrée dans le codeur). Les symboles en sortie sont donc déterminés à partir de l'état du codeur et du bit en entrée. On utilise cette propriété pour élaborer un autre diagramme dénommé le treillis tenant compte de cette structure répétitive.

- Le treillis

On utilise pour le construire la notion d'état définie précédemment. Les deux chemins issus de chaque état correspondent à l'entrée dans le codeur d'un "1" ou d'un "0". Cette entrée qui produit certains symboles en sortie fait également évoluer le codeur vers un nouvel état. Ce schéma, s'il présente une forme plus compacte que l'arbre, n'est cependant pas extrêmement lisible. Une forme plus synthétique pour représenter l'évolution du système est le diagramme d'états.

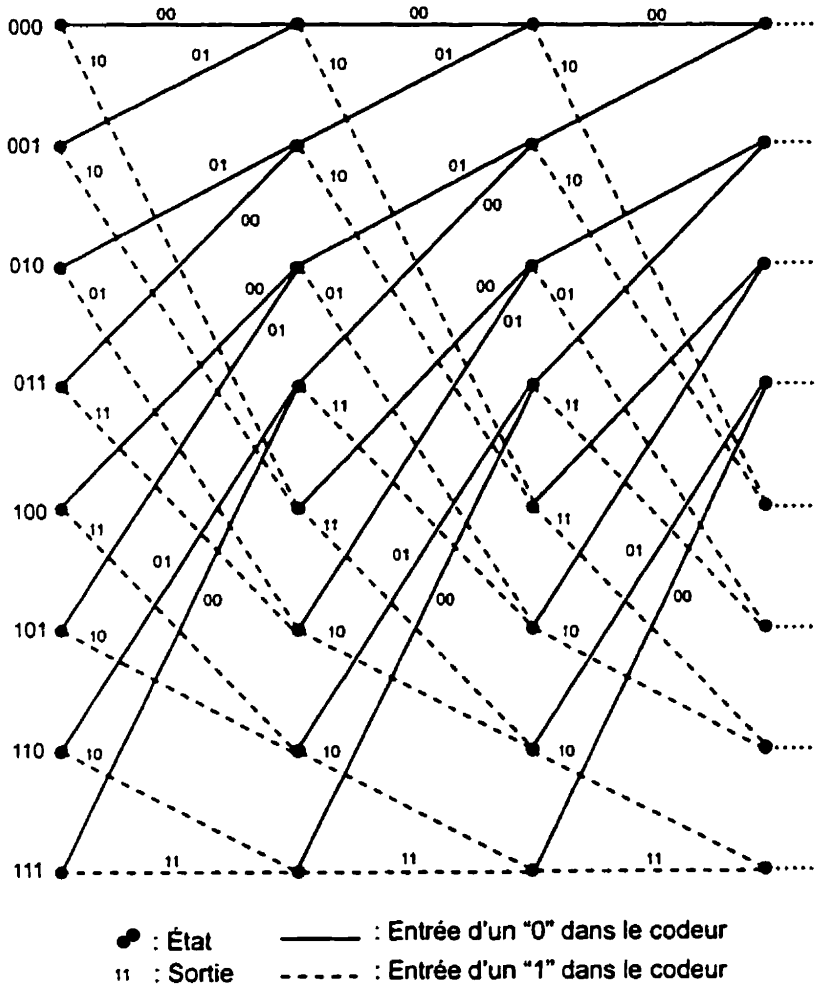

Figure 2.6 - *Treillis correspondant au codeur de taux* $\frac{1}{2}$, $K = 4$, *présenté à la figure 2.3*

- Le diagramme d'états

Celui-ci regroupe de façon simple et lisible les différentes états dans lesquels peut se trouver le codeur et les transitions possibles entre ceux-ci.

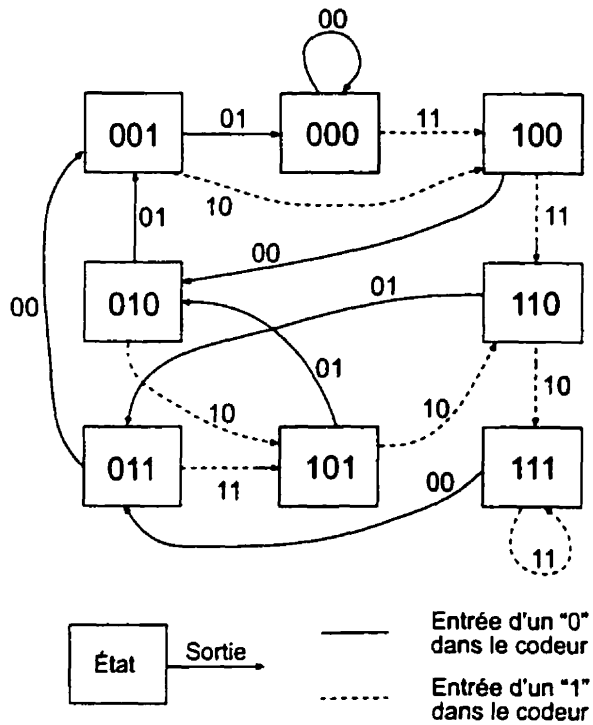

Figure 2.7 – *Diagramme d'états correspondant au codeur de taux* $\frac{1}{2}$, $K = 4$, *présenté à la figure 2.3*

L'une ou l'autre des deux premières structures (figures 2.5 et 2.6) est utilisée par les algorithmes de décodage pour corriger d'éventuelles erreurs survenues lors de la transmission.

2.2.2.3 Algorithmes de décodage

Deux grandes classes de décodage sont utilisées : le décodage à seuil et le décodage à maximum de vraisemblance (connu sous le nom de son inventeur A. J. Viterbi). Ce dernier est optimal dans le sens où il permet de minimiser la probabilité d'erreur par séquences. J. L. Massey [1] qui introduisit une grande partie de ces méthodes de décodage montra que l'information de vraisemblance pouvait être rendue optimale en ajustant de façon judicieuse les facteurs de poids sur l'ensemble des seuils des

valeurs d'entrées. Il donna ainsi naissance au décodage à seuil qui présente le gros avantage d'être très peu complexe notamment par rapport à celui de Viterbi. Le principe du décodage à seuil sera décrit au chapitre suivant (3.2.2.1).

La procédure de Viterbi

L'algorithme de décodage de Viterbi est une procédure fort efficace principalement pour des codes binaires ayant une faible longueur de contrainte ($K \leq 9$). Il s'agit d'une procédure de recherche à travers l'arbre ou le treillis de la séquence la plus probable selon une métrique Euclidienne ("soft decision") ou de Hamming ("hard decision"). Plus précisément, la métrique de la *jième* branche du *ième* chemin au sein du treillis est définie comme le logarithme de la probabilité d'occurrence de la séquence reçue $Y_j = \{y_{j,m}\}$ (où m est le *mième* bit de la branche j) conditionnellement à la séquence transmise $C_j^{(i)} = \{c_{jm}^{(i)}\}$ pour le *ième* chemin. Soit : $\gamma_j^{(j)} = log(P(Y_j|C_j^{(i)}))$, j = 1,2,3,.... Ainsi une métrique pour le *ième* chemin constitué de B branches à travers le treillis est défini par : $\Gamma^{(i)} = \sum_{j=1}^{B} \gamma_j^{(i)}$. Le choix entre deux chemins différents du treillis s'effectue en optant pour celui disposant de la métrique la plus importante. Cette règle permet de maximiser la probabilité d'avoir pris une bonne décision ou (ce qui est équivalent) de minimiser la probabilité d'erreur pour la séquence de bits d'information en entrée.

Utilisation :

On s'intéresse à un noeud particulier (état) et plus précisément à tous les chemins aboutissant à celui ci. On ne retient que le chemin (nommé survivant) disposant de la plus faible métrique et on élimine tous les autres. On effectue cela pour tous les noeuds d'un même niveau et ceci pour tous les niveaux. Il ne reste plus alors qu'à déterminer le chemin global correspondant à la séquence d'entrée

en identifiant à travers les survivants le parcours ayant la plus faible métrique.

De manière générale lorsque l'on considère un code convolutionnel binaire avec $v = 1$ et une longueur de contrainte K il y a 2^{K-1} états. Ainsi il y a 2^{K-1} chemins survivant à chaque niveau et 2^{K-1} métriques (une pour chaque survivant). Si l'on suppose un système où $v \neq 1$ on aboutit à un treillis avec 2^{K-v} états. À chaque niveau du treillis il y a donc 2^v chemins qui aboutissent à chaque noeud et qui sont issus de chaque noeud. On voit donc apparaître là une limitation importante de cette procédure: le nombre de calculs effectués au décodage à chaque niveau du treillis augmente de manière exponentielle avec v et K ce qui restreint le champ d'application de cet algorithme à de faibles valeurs de ces deux grandeurs.

2.2.2.4 Avantages et inconvénients

a) Avantages

Le principal avantage des codes convolutionnels est sans nul doute la facilité de mise en oeuvre. De plus les codes convolutionnels délivrent de très bonnes performances d'erreur (les meilleures à l'heure actuelle).

b) Inconvénients

Leurs inconvénients résident principalement dans l'utilisation du décodeur de Viterbi. En effet, si ce dernier est à la base de la qualité des performances de ce type de codage, il présente le désavantage ou plus exactement la limitation suivante: l'augmentation de la longueur de contrainte qui aboutit à une augmentation du gain de codage et donc à une amélioration des performances d'erreur, entraîne un accroissement de la complexité du décodage suivant une loi exponentielle. Ce fait est tel qu'à partir d'une certaine longueur de contrainte (usuellement $K = 9$), l'implémentation devient extrêmement difficile.

2.2.2.5 Évolutions

Les codes convolutionnels ne présentant pas que des avantages, la recherche dans le domaine du codage a connu par la suite de nombreuses tentatives visant à construire des codes puissants dont le décodage serait élaboré de sorte à subdiviser l'étape de décodage en une succession d'étapes plus simples de décodage partiel. Notons entre autres l'apparition des codes produits [36], des codes concaténés [3], des codes multi-niveaux [37] ainsi que différentes techniques de décodage (décodage de Viterbi adaptatif [24], décodage bidirectionnel [25], ...). Il fallut cependant attendre 1993 et l'introduction du codage turbo pour voir apparaître une profonde innovation : l'échange d'information entre les différentes parties composant le décodeur.

2.3 La technique du codage turbo

2.3.1 Historique

Le codage turbo est une technique récente. Elle fut présentée pour la première fois en 1993 lors de la conférence I.C.C'93 ("International Conference on Communications") à Genève par C. Berrou [22] Directeur d'études à l'école Nationale Supérieure des Télécommunications de Bretagne (E.N.S.T.B) en France. Dans sa présentation celui ci considérait le décodage itératif de deux codes convolutionnels systématiques récursifs concaténés en parallèle par l'intermédiaire d'un entrelaceur non uniforme. Le décodage était assuré par un décodeur "entrées et sorties pondérées" ("soft input/soft output") utilisant un algorithme de maximum de vraisemblance. Cette découverte constituait une des évolutions les plus importantes dans le domaine du codage depuis l'introduction de la notion de "treillis-coded modulation" par Ungerboeck [31] en 1982. Les performances de ces codes sont en effet exceptionnellement voisines de la capacité du canal.

En 1994, R. Pyndiah [38] également professeur à l'E.N.S.T.B proposa un codage turbo basé sur les codes en blocs à la conférence "Globecom'94" de San Fransisco. Sa technique reposait sur le décodage itératif de deux codes B.C.H (Bose-Chaudhuri-Hocquenghem [28], [29] et [30]) concaténés en série à travers un entrelaceur uniforme. Pour décoder les composantes du code R. Pyndia proposa un nouveau décodeur "soft input/soft output" pour les codes en blocs.

Ces deux types de codes turbo sont relativement distincts, ils utilisent en effet différents schémas de concaténation et différents algorithmes "soft input/soft output". Les codes turbo basés sur les codes convolutionnels sont communément désignés sous l'appellation C.T.C ("convolutional turbo codes"), ceux basés sur les codes en blocs héritent de l'appellation B.T.C ("block turbo codes").

Depuis 1993 et la parution du premier article sur le sujet, il y a eu de très nombreuses publications et beaucoup de recherches (les principales seront décrites par la suite) ont été entreprises dans ce domaine. Au début pourtant, les concepts novateurs et compliqués auxquels faisaient appel cette technique firent obstacle à son étude. Il fallut de plus vaincre le scepticisme qui accueillit cette découverte aux performances remarquables. En 1997 l'E.N.S.T.B organisa le premier colloque sur le sujet des codes turbo.

De nos jours ce type de codes est considéré comme le plus efficace dans la catégorie F.E.C ("forward error correction") et est implémenté dans de nombreuses applications. Il fait d'ailleurs partie des nouvelles normes pour les futurs systèmes cellulaires CDMA. Nous aurons l'occasion dans les parties subséquentes de préciser les concepts évoqués ci-dessus et de présenter les avantages et les inconvénients de ce type de codage.

2.3.2 Description

L'historique précédent a déjà donné un premier aperçu de la structure du codage turbo. Celle-ci consiste au niveau de l'encodeur en la concaténation parallèle de deux codeurs récursifs et systématiques à travers un entrelaceur. Le décodage est effectué de façon itérative par deux décodeurs concaténés en série.

a) L'encodeur

Sa structure générale est reproduite à la figure 2.8.

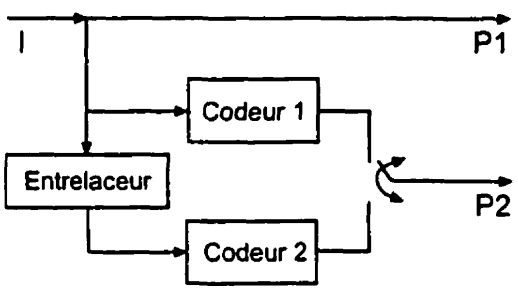

Figure 2.8 - *Codage turbo*

Le fait que les deux codeurs opèrent sur le même ensemble de bits qui apparaît en parallèle au lieu d'agir l'un à la suite de l'autre justifie l'appellation "parallèle". La présence de l'entrelaceur est primordiale. Celui ci permet en permutant l'ordre des bits de la séquence d'entrée de fournir au deuxième codeur une séquence statistiquement indépendante de celle qui est traitée par l'autre codeur. Ces codeurs, qui peuvent être en nombre supérieur à deux, sont quasiment toujours choisis systématiques et récursifs.

b) Le décodeur

Le décodage turbo s'effectue suivant un mode modulaire de façon séquentielle. Du fait de la concaténation au niveau de l'encodeur, l'objectif du décodeur turbo est de décomposer le processus de décodage en plusieurs étapes simples afin de réduire

la complexité du système global. Le schéma présenté figure 2.9 illustre l'idée de décodeur proposée par Berrou, Glavieux et Thitimajshima [22] en 1993.

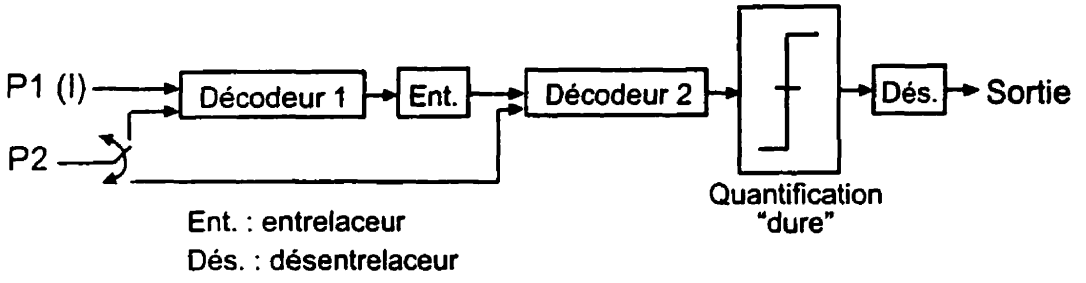

Figure 2.9 – *Décodage turbo*

On remarque ici que la solution proposée est celle d'une concaténation série contrairement au type de concaténation adoptée au niveau de l'encodeur. Ceci est cohérent si l'on se rappelle que l'encodeur agit sur la même séquence d'entrée. Pour simplifier on peut dire que l'information est encodée deux fois au niveau de l'encodeur et donc qu'il est normal que la séquence reçue soit décodée deux fois ce que seule une architecture série permet. Plus précisément, le premier décodeur reçoit les symboles issus de la séquence d'information (symboles systématiques) et ceux issus de la sortie du premier codeur de l'encodeur (symboles de parité). Le rôle du premier décodeur est de fournir pour chaque symbole qu'il a décodé une estimation de la fiabilité du choix effectué. La sortie du premier décodeur est alors passée à travers un entrelaceur de structure identique à celle de l'entrelaceur utilisé pour l'encodage. Le deuxième décodeur dispose donc des résultats obtenus par le premier décodeur et des symboles de parité générés par le deuxième encodeur. Un algorithme est alors utilisé pour obtenir une séquence d'information décodée et remise dans l'ordre.

Le fonctionnement ainsi présenté ne permet pas d'obtenir un décodage global optimal car le premier décodeur ne prend en compte que la moitié des symboles de parité reçus. L'ajout d'une boucle de rétroaction comme cela est illustré à la

figure 2.10 permet d'apporter une modification importante induisant ainsi la notion de fonctionnement itératif qui fait la "force" de ce type de codage. Le deuxième décodeur choisi dans ce cas doit, tout comme le premier, générer une information de fiabilité (utilisation d'un algorithme de décodage de symboles). Cette dernière est alors passée dans le délaceur afin d'être utilisée comme *information a priori* par le premier décodeur à la prochaine itération. Ainsi, dès la seconde itération, le premier décodeur dispose non plus de deux mais de trois entrées. Les performances du système peuvent alors s'améliorer de façon significative d'itération en itération. Au bout d'un nombre fixé d'itérations, la sortie du deuxième décodeur est quantifiée et une estimation de la séquence transmise est délivrée.

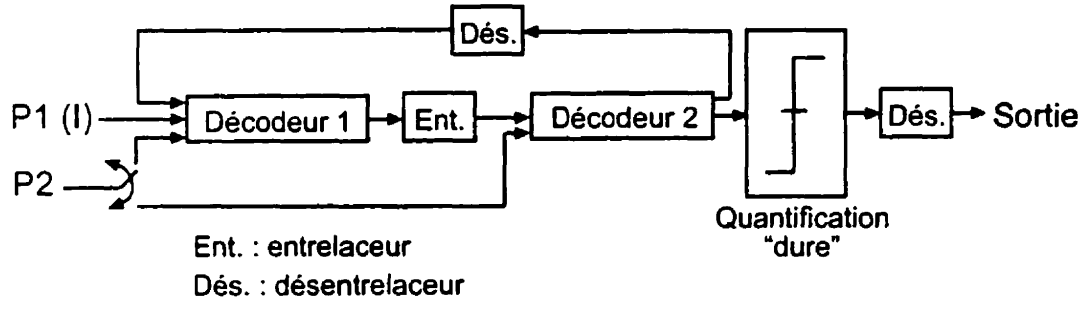

Figure 2.10 – *Décodage turbo avec boucle de rétroaction*

c) L'algorithme

À la vue du principe de fonctionnement du décodage énoncé, l'algorithme utilisé doit être capable de fournir une information de fiabilité pour chaque symbole décodé. Cette information est dénommée probabilité a posteriori (PAP). L'algorithme de Viterbi (2.2.2.3) principalement utilisé pour le décodage des codes convolutionnels maximise de façon optimale la probabilité a posteriori par séquence (sans garantir toutefois l'optimalité par symbole). Les travaux de Chang et Hancock [35] ont donné naissance à une procédure minimisant la probabilité d'erreur par symbole (MAP). Quelques années plus tard, Bahl et Al [26] ont adapté cette technique

aux codes correcteurs d'erreurs. C'est cet algorithme qui fut implémenté pour le codage turbo.

2.3.3 Avantages

L'encodage turbo est le type de codage le plus puissant au chapitre des performances à l'heure actuelle. Lors de sa présentation, une probabilité d'erreur par bit de $P_b = 10^{-5}$ était annoncée pour un rapport signal à bruit de $\frac{E_b}{N_0} = 0.7dB$ soit à 0.7 dB de la capacité du canal (résultats obtenus avec deux codeurs de 16 états, un entrelaceur de 65536 bits et un décodage en 18 itérations utilisant l'algorithme MAP). Ces performances qui sont initialement demeurées mystérieuses du fait de l'absence d'explications théoriques, sont principalement dues à la combinaison de deux faits essentiels: l'algorithme utilisé pour le décodage et les éléments constituant le codeur.

Quelques caractéristiques remarquables de la qualité des codes turbo sont présentées ci-dessous. Ceci ne constitue naturellement pas une classification exhaustive et on se référera à des ouvrages spécialisés [49], [50] pour de plus amples précisions.

- Pour des longueurs de blocs inférieures à 1000 bits il n'est pas nécessaire d'aller plus loin que 6 itérations pour un rapport signal à bruit compris entre 1.5 et 3 dB.
- Pour des codes de taux $\frac{1}{2}$ et $\frac{1}{3}$ la supériorité du codage turbo sur les autres méthodes est clairement établie.
- Même dans des canaux très bruités le codage turbo demeure très efficace.
- Il existe de nombreuses façons d'agir sur les performances des codes turbo. Citons les principaux paramètres d'influence:
  - la longueur de l'entrelaceur,
  - le type d'entrelacement,

- la longueur de contrainte des codes élémentaires,
- les générateurs des codes choisis à l'encodeur,
- le taux de codage.

Cette flexibilité des paramètres est un avantage indéniable pour affiner le codage turbo selon les propriétés recherchées. Cela rend en contrepartie son étude plus délicate.

2.3.4 Inconvénients

On discerne principalement deux grands inconvénients qui sont autant de limitations pour certaines applications :

a) La complexité de l'algorithme de décodage

La version initiale de l'algorithme utilisé faisait apparaître une grande complexité et la nécessité de disposer d'une importante place mémoire et de devoir effectuer de gros calculs. D'autre part la structure itérative retenue induit un délai proportionnel à la longueur de l'entrelaceur et au nombre d'itérations effectuées. En règle générale plus ce dernier nombre sera important et plus la performance du système sera améliorée. Une des façons efficaces de lutter contre le délai ainsi introduit est de restreindre l'application des codes turbo à des séquences dont la longueur n'excède pas 1000 bits. Il est également possible de faire un compromis en réduisant le nombre d'itérations effectuées au détriment des performances.

b) La complexité matérielle

On aura remarqué que la réalisation matérielle du codage turbo n'est pas des plus simples. La présence de deux encodeurs, de deux décodeurs et de deux entrelaceurs en fait un système plus complexe que d'autres architectures existantes. De plus, la mise en œuvre de l'algorithme MAP est relativement compliquée.

c) La complexité théorique

L'étude du codage turbo de manière analytique s'avère très complexe. La difficulté principale se situe au niveau de l'étude analytique de la concaténation parallèle des codes à travers l'entrelaceur. Ces difficultés théoriques ont poussé les chercheurs à se baser principalement au départ sur des résultats de simulations pour l'analyse des performances. Ceci, on s'en doute, n'était pas forcément la méthodologie la plus adaptée à l'interprétation de certains comportements caractéristiques de cette classe de codes. Il y a cependant eu depuis de très bonnes analyses théoriques [6].

d) La "saturation" des performances

Lorsque l'on observe le comportement des performances des codes turbo sur une plage de rapport signal à bruit suffisamment grande on s'aperçoit qu'à partir d'un certain $\frac{Eb}{No}$, les courbes d'erreur tendent vers une asymptote "faiblement inclinée", ceci indépendamment du choix des paramètres du système. Une explication rigoureuse de ce phénomène fut fournie par Perez et Al [39].

2.3.5 Évolutions

Le potentiel des codes turbo est tel que de nombreuses recherches ont été entreprises pour supprimer ou tout du moins pour réduire leurs limitations. Toutes se sont astreintes à étudier un point particulier de la structure connue. On notera principalement des améliorations envisagées au niveau de:

a) L'algorithme de décodage

C'est sans doute là que se sont concentrés le plus d'efforts. L'idée étant d'optimiser la probabilité d'erreur par symbole (ce que ne fait pas l'algorithme de Viterbi) et également de réduire la complexité mise en jeu lors du décodage. Dans cette catégorie nous pouvons citer le travail de Robertson [40] sur une simplification de

l'algorithme MAP reprise par la suite par Berrou [10]. Une amélioration importante fut introduite avec l'expression de la procédure MAP dans le domaine logarithmique (log-MAP) [41], [42] et [43]. On peut également évoquer les études de J. Hagenauer [33], [34] pour l'utilisation d'une amélioration de l'algorithme de Viterbi connue sous le nom de S.O.V.A [32] (soft output Viterbi algorithm).

b) La structure de l'entrelaceur

Pièce essentielle de la structure turbo, l'entrelaceur a été également l'objet de nombreuses recherches. Diverses structures ont été envisagées. On peut répartir celles-ci en deux grandes classes :

- *les entrelaceurs déterministes :* on citera comme exemples les entrelaceurs blocs ou hélicoïdaux. Dans ce type de structure la connaissance de la place d'un bit dans un bloc suffit à déterminer sa position après entrelacement du bloc. Ceux ci s'avèrent efficaces dans le cas de séquences courtes.
- *les entrelaceurs pseudo-aléatoires :* chaque bit dans une séquence à entrelacer se voit affecter un nombre aléatoire qui correspond à sa place dans le bloc après entrelacement. Ceux-ci sont dans la majorité des cas la meilleure option pour des longueurs de séquences supérieures à 400.

D'une manière générale il convient d'effectuer un compromis au niveau de la longueur de l'entrelaceur. En effet plus celle ci sera importante et meilleures seront les performances au détriment cependant du délai engendré. On se référera au mémoire de G. Royer [7] pour de plus amples précisions.

c) Le type de codes utilisés dans la concaténation de l'encodeur

Outre le fait qu'il soit indispensable d'opter pour des codes systématiques et récursifs il est important, pour obtenir de meilleures performances, d'augmenter la distance libre (d_{libre}) du code.

2.3.6 Conclusion

Ainsi, au fil des années, les nombreux travaux et analyses ont permis de rendre les codes turbo encore plus performants et ont élargi leurs champs d'applications. Il n'en demeure pas moins que ceux-ci conservent quelques limitations et s'avèrent inadaptés à certaines applications. Il est alors dommage de ne pas pouvoir bénéficier de la "qualité turbo" dans ces situations. Ce constat est à l'origine des recherches entreprises par F. Gagnon et D. Haccoun qui ont abouti à une évolution importante [12]. Celle-ci précédemment évoquée est détaillée dans le chapitre suivant.

CHAPITRE 3

CODAGE DOUBLEMENT ORTHOGONAL

3.1 Orthogonalité, considérations générales

Il serait vain d'évoquer le concept d'orthogonalité dans sa globalité tant les définitions et les applications varient suivant les domaines. Cependant il apparaît intéressant de souligner l'apport de la propriété d'orthogonalité à la chaîne traditionnelle de communications.

D'un point de vue "télécommunications", l'orthogonalité entre deux signaux permet d'obtenir une non-corrélation et donc une non-interaction entre les deux entités. Ce fait est primordial puisqu'il donne la possibilité de réduire le problème des "interférences". Ainsi on peut citer sans être exhaustif, l'utilisation du principe d'orthogonalité pour :

- l'affectation de fréquences,
- la répartition d'accès,
- la modulation et la démodulation,
- l'estimation (principe d'orthogonalité dans l'estimation de l'erreur quadratique moyenne),
- le codage et le décodage.

Le dernier point est celui qui nous préoccupe dans le cadre de notre étude et que nous allons décrire plus en détails par la suite.

3.2 Rappels sur les codes simplement orthogonaux

3.2.1 Historique

Les codes simplement orthogonaux ont été introduits pour la première fois par J.L. Massey [1]. Ils sont nés de la recherche d'une technique permettant de corriger les erreurs de transmission de façon peu coûteuse et simple. Celui-ci proposa une procédure pour déterminer de tels codes avec t (nombre arbitraire) symboles de syndromes dans chaque ensemble orthogonal et un taux de codage de R. Il montra que tous les codes simplement orthogonaux pouvaient être orthogonalisés de telle sorte à ce que l'on ait : $d_{min} = t + 1$ et étaient donc décodables par une procédure à seuil. S'il se concentra principalement sur de faibles taux de codage, les travaux de J. Robinson [45] vinrent compléter son étude. L'importance de trouver des codes de taux élevés n'est en effet pas à négliger car la méthode de décodage à seuil adoptée est très peu complexe et très facile d'implémentation ce qui produit pour des codes de forts taux une différence notable avec les anciennes méthodes utilisées. De plus, l'évolution technologique constante au niveau des architectures V.L.S.I (intégration à très grande échelle) repousse sans cesse la longueur maximale de contrainte "acceptable".

3.2.2 Définition

L'orthogonalité au sein des algorithmes de décodage se rapporte à l'utilisation d'un ensemble de séquences de syndromes pour corriger un symbole erroné durant la transmission. Le bit correspondant à ce symbole erroné est injecté dans chaque séquence de syndromes de l'ensemble orthogonal de sorte à ce qu'aucun autre bit ne soit vérifié par plus d'une séquence de ce type dans l'ensemble considéré.

Un code convolutionnel est par définition appelé simplement orthogonal si et

seulement si l'ensemble des symboles de syndromes vérifiant chaque symbole d'erreur forme un ensemble d'équations de parité qui sont orthogonales sur ce symbole (on ne considérera que des alphabets binaires).

Illustrons notre propos en considérant un exemple inspiré de celui fourni par J.L. Massey [1] lui même (les notations seront les mêmes que celles utilisées en 2.2.2.2 avec $G = (g_i)$ sous forme systématique).

Soient $(i_0, i_1, ..., i_{k-1})$ quelques bits de notre séquence d'information en entrée. Soit $(c_0, ..., c_{k-1}, c_k, ..., c_{n-1})$ l'ensemble des symboles codés transmis correspondants. On considérera le cas d'un code systématique : $(c_0, ..., c_{k-1}) = (i_0, ..., i_{k-1})$, $(c_k, ..., c_{n-1})$ sont les symboles de parité rajoutés à des fins de vérification. Leur expression est du type : $c_j = \sum_{l=0}^{k-1} g_{j,l}.i_l$ pour $j = k, ..., n-1$. Soient $(y_0, ..., y_{n-1})$ les symboles reçus. Ceux-ci à cause des erreurs de transmission diffèrent des signaux transmis. On note $(e_0, ..., e_{n-1})$ la séquence d'erreur $(y_i = c_i + e_i)$. On peut définir une séquence de vérification de parité à la réception par l'intermédiaire des syndrômes : $s_j = \sum_{i=0}^{k-1} g_{j,i}.y_i - y_j$ ou encore $s_j = \sum_{i=0}^{k-1} g_{j,i}.e_i - e_j$. La procédure qui est exposée ci-après consiste à transformer les symboles vérificateurs de parité $\{s_j\}$ en un autre ensemble simplifiant la procédure de décodage. On définit ainsi une séquence de vérification de parité A_i, combinaison linéaire $(a_{i,j})$ des $\{e_j\}$ par : $A_i = \sum_{j=0}^{n-1} a_{i,j}.e_j$ $(i = 0, 1, ..., t-1)$. Un ensemble de t séquences de vérification de parité $\{A_i\}$ est dit orthogonal vis à vis du symbole d'erreur e_m si :

$$\begin{cases} a_{i,m} = 1,\ i = 0, 1, ..., t-1 \\ a_{i,j} = 0\ \forall\ i\ (\textit{sauf pour au plus une valeur de } i)\ \textit{pour } j \neq m\ \textit{fixé}. \end{cases}$$

On peut résumer l'expression précédente en stipulant qu'un ensemble de t séquences de vérification de parité est dit orthogonal vis à vis de $\{e_m\}$ si chaque expression de vérification porte sur e_m et s'il n'existe aucun autre symbole e_i ap-

paraissant dans plus d'une équation de vérification.

Considérons un exemple simple binaire de taux de codage $R = \frac{3}{5}$:
information : (i_0, i_1, i_2)
signal transmis : $(c_0, c_1, c_2, c_3, c_4)$ avec $c_0 = i_0$, $c_1 = i_1$, $c_2 = i_2$, $c_3 = i_0 + i_1 + i_2$ et $c_4 = i_0 + i_1$
Soit en reprenant les notations précédentes :
$g_{3,0} = 1$, $g_{3,1} = 1$, $g_{3,2} = 1$; $g_{4,0} = 1$, $g_{4,1} = 1$ et $g_{4,2} = 0$.
Considérons une orthogonalité définie sur le symbole e_2. On a donc : $a_{i,2} = 1\ \forall i$ et $a_{i,j} = 0\ \forall i$ (une unique exception est tolérée) pour $j \neq 2$. On choisit à titre d'exemple les combinaisons suivantes :
$A_0 = e_0 + e_2$ $(a_{0,0} = 1, a_{0,2} = 1)$
$A_1 = e_1 + e_2 + e_3$ $(a_{1,1} = 1, a_{1,2} = 1, a_{1,3} = 1)$
$A_2 = e_2$ $(a_{2,2} = 1)$
$A_3 = e_2 + e_4$ $(a_{3,2} = 1, a_{3,4} = 1)$
$A_4 = e_2$ $(a_{4,2} = 1)$
On vérifie que mis à part e_2 qui est présent dans chacune des équations, aucun symbole d'erreur n'apparaît plus d'une fois dans les équations.

3.2.2.1 Décodage à seuil

Une fois cela établi il est possible de décrire différents algorithmes pouvant être utilisés pour déterminer e_m à partir d'un ensemble de t équations de parité A_m orthogonales sur e_m [1]. On formule l'hypothèse d'un bruit blanc gaussien.

a) Décodage à la majorité

Dans l'hypothèse où le nombre d'erreurs au sein des symboles reçus est inférieur à $\lfloor \frac{t}{2} \rfloor$, la valeur de e_m sera correctement déterminée en choisissant celle qui permet

de vérifier le plus grand nombre d'équations A_i.

Le type de décodage présenté ci dessus revêt le nom de "décodage à la majorité". On peut le rendre plus efficace en tenant compte de la "forme" du canal et donc de la modélisation statistique associée. On utilise alors la dénomination de décodage à "probabilité a posteriori".

b) Décodage à probabilité a posteriori

En considérant un modèle de bruit blanc et additif, la valeur V de e_m sera choisie de sorte à ce que l'expression $log[Pr(e_m = V)] + \sum_{i=0}^{t-1} log[Pr(A_i|e_m = V)]$ soit maximale.

c) Décodage à seuil

Si l'on considère la spécialisation des techniques précédentes au cas binaire (qui est celui qui nous préoccupe dans le cadre de notre étude), on obtient :

Règle de décodage à la majorité :

On choisit e_m égal à "1" si et seulement si la somme des $\{A_i\}$ excède la valeur seuil fixée à $\lceil \frac{t}{2} \rceil$. Ainsi si $\sum_{i=0}^{t-1} A_i > \lceil \frac{t}{2} \rceil$ on décide alors que le symbole e_m est un "1". Dans le cas contraire, un "0" est sélectionné.

Règle de décodage à probabilité a posteriori :

On choisit e_m égal à "1" si et seulement si la somme de tous les éléments de l'ensemble A_i des t équations de parité orthogonales sur e_m pondérée par le facteur $2.log(\frac{q_i}{p_i})$ dépasse la valeur de seuil $\sum_{i=0}^{t} log(\frac{q_i}{p_i})$ où $p_0 = 1 - q_0 = Pr(e_m = 1)$ et $p_i = 1 - q_i$ est la probabilité d'obtenir un nombre impair d'erreurs au sein des

symboles testés par la *ième* équation de parité.

Ces deux règles étant relativement similaires on utilise le même terme de "décodage à seuil" pour les décrire.

3.2.2.2 Codes orthogonaux

Maintenant que la propriété d'orthogonalité a été explicitée au niveau du décodage, il est possible d'introduire la notion de codes orthogonaux. On dit simplement qu'un code est orthogonal si l'ensemble des séquences de syndromes qui teste e_i est également un ensemble d'équations de vérification orthogonales sur e_i.

Pour exemple de code simplement orthogonal on peut citer le code $\frac{8}{9}$ utilisé pour un système T.D.M.A (accès multiple à répartition dans le temps) dont le taux de codage est choisi de sorte à faire correspondre chaque échantillon P.C.M (pulse code modulation) à 8 bits par bloc. Les générateurs de ce code sont reproduits dans le tableau 3.1 sous forme de notation matricielle (les trois nombres entre parenthèses représentent la position des trois connexions entre le registre et l'additionneur de la séquence de sortie considérée).

Tableau 3.1 - *Exemple de code simplement orthogonal de taux* $\frac{8}{9}$ *utilisé pour un système T.D.M.A*

parité	générateurs
P_1	(38,56,68)
P_2	(32,43,53)
P_3	(5,13,50)
P_4	(9,49,136)
P_5	(48,52,72)
P_6	(71,81,82)
P_7	(2,15,35)
P_8	(6,28,57)

W. W. Wu [46] répertorie de nombreux codes orthogonaux pour différents taux de codage (de $\frac{2}{3}$ à $\frac{13}{14}$) avec un nombre de générateurs variant de 3 à 50 obtenus à partir d'une méthode basée sur l'étude de triangles.

3.2.3 Propriétés - utilisations

La technique du codage simplement orthogonal possède les avantages suivants:

- une implémentation simple,
- pas de propagation d'erreur,
- une capacité de correction garantie au delà de la distance minimale,
- la capacité d'opérer les opérations de codage et de décodage très rapidement,
- un grand nombre de codes possibles.

On a déja évoqué l'application de ce type de codes au domaine spatial. Pour citer quelques exemples il est possible de parler du système INTELSAT dont le "codec" (système SPADE) utilisait ce type de code avec un taux $R = \frac{3}{4}$, des satellites ANIK canadiens et de son intégration à différents systèmes TDMA.

3.3 Origine du codage doublement orthogonal

3.3.1 Insuffisances du codage turbo

On a déjà eu l'occasion d'évoquer les inconvénients que présentent le système de codage turbo. On peut les regrouper en deux groupes:

a) La complexité

Celle ci se retrouve à plusieurs niveaux:

- complexité matérielle du fait de l'utilisation d'entrelaceurs et d'au moins deux codeurs et de deux décodeurs. Ce point entraîne également un coût

élevé de mise en œuvre.

- complexité théorique engendrée par la difficulté d'analyser le comportement de l'entrelaceur,
- complexité algorithmique au niveau du décodage.

b) Le délai

On différencie là aussi plusieurs facteurs d'influence:

- la latence au décodage due à la taille des entrelaceurs (plus cette dernière grandeur est élevée et meilleures seront les performances),
- le nombre d'itérations qui en augmentant améliore les probabilités d'erreur.

Afin d'élargir le champ d'applications de la "qualité turbo", il apparaît inévitable de devoir remédier à ces insuffisances. C'est dans cette optique que fut introduite l'évolution importante qui est présentée à la section suivante.

3.3.2 Amélioration radicale envisagée

3.3.2.1 Idée

Le but poursuivi est tel qu'améliorer chaque composante indépendamment comme cela a été le cas jusqu'à présent (2.2.2.5) ne sera pas suffisant. Une solution plus radicale visant à réduire la complexité et la latence d'un système turbo fut proposée par C. Cardinal, D. Haccoun, F. Gagnon et N. Batani [12] en 1997 et a fait l'objet d'un dépôt de brevet [5].

L'approche "audacieuse" est:

- réduction de la complexité → adoption d'une simple procédure de décodage à seuil,
- réduction de la latence → suppression des entrelaceurs,

- maintien d'un bon niveau de performances d'erreur → conservation de la structure itérative.

Sachant que l'entrelaceur est une pièce essentielle du codage turbo comment peut-on espérer conserver la "qualité turbo" en se privant de cet élément? L'idée adoptée par les inventeurs consiste à reporter la complexité matérielle au niveau des codes utilisés (i.e.: les générateurs doivent posséder certaines propriétés particulières).

L'attrait d'un tel système théorique apparaît immédiatement, dès lors que l'entrelaceur n'existe plus ni au niveau de l'encodage, ni au niveau du décodage il n'est plus d'aucune utilité de disposer de deux codeurs et de deux décodeurs. On aboutit alors à un système de codage des plus simples donc facile à mettre en œuvre, peu onéreux et a priori sensiblement plus rapide.

3.3.2.2 Mise en œuvre

Il convient d'être très prudent, on se doute bien en effet qu'aboutir à la "qualité turbo" tout en supprimant une partie de la structure de ce type de codage ne se fera pas sans difficultés. Afin d'espérer atteindre le même niveau de performances il est indispensable de conserver une structure itérative au décodeur comme cela est le cas pour les codes turbo. Afin de bénéficier pleinement de cette implémentation itérative il est primordial de garantir l'indépendance entre les observables à chaque itération.

Ces conditions exposées il reste à déterminer les propriétés que doivent vérifier les codes de sorte à ce qu'à chaque itération les entrées du sytème soient non corrélées.

3.3.2.3 Travaux précédemment réalisés

Cette technique novatrice ne date que de quelques années. Les inventeurs ont tout d'abord définit et formalisé la propriété de double orthogonalité qui satisfait les principes précédents. Ils ont ensuite effectué différents travaux théoriques et pratiques pour générer ce type de codes. Cette nouvelle classe de codes appelés CSO^2C (pour "convolutional self-doubly orthogonal codes") ont donné des résultats très encourageants lors des premières simulations même si la longueur des codes obtenus ne donnait pas complète satisfaction.

Ci-après figure le raisonnement suivi par M. Gagnon et M. Haccoun [12] qui a permis d'aboutir à la condition de double orthogonalité pour les codes utilisés dans le cas d'une structure sans entrelaceur.

On considérera pour simplifier la démonstration un code de taux $\frac{1}{2}$. Ceci ne restreint en rien le champ d'applications de cette démonstration. Un encodeur systématique sera choisi comme cela était le cas pour les codes turbo.

Chaque symbole de parité est obtenu comme la somme modulo 2 de J bits de la séquence d'information. La position de ces J bits au sein de la séquence est définie par l'ensemble des J générateurs utilisés. Dans notre exemple schématisé par la figure 3.1, trois générateurs sont utilisés aux positions (0, 2, 5).

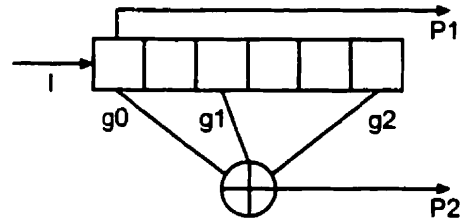

Figure 3.1 – *Illustration de l'exemple de code de taux* $\frac{1}{2}$ *considéré*

On peut donc modéliser le système par l'ensemble de $J = 3$ éléments $\{g_0, g_1, g_2\}$ où $g_0 = 0$, $g_1 = 2$ et $g_2 = 5$ représentent la position des connexions entre le registre à décalage et l'additionneur. Par exemple, $g_0 = 0$ désigne la case du registre contenant le bit d'information. La séquence d'information sera notée I. La notation I^k désignera le bit de la séquence d'information à l'entrée du système à l'instant k. Les séquences de parité seront désignées par P_i (où i est la séquence considérée), la même notation que pour les bits d'information P_i^k sera utilisée. Ainsi les équations de parité s'écrivent :

$$P_1^k = I^k, \tag{3.1}$$

$$P_2^k = \sum_{i=0}^{J-1} I^{k-g_i}. \tag{3.2}$$

Au niveau du décodage, en inversant une équation de parité correspondant à un symbole d'information particulier, il est possible d'obtenir une estimation de ce symbole. Il est à noter que la valeur de J définit les capacités de détection et de correction du code (pour les codes orthogonaux, la distance minimale est égale à $J + 1$). Pour un symbole d'information particulier, il est possible d'obtenir $J + 1$ estimées en inversant les équations de parité. Nommons $\hat{I}$ et $\hat{P}$ les grandeurs reçues (respectivement la séquence d'information et les équations de parité). Le décodage à seuil prend, à partir des $J+1$ estimations des symboles d'information, une décision

pour déterminer quel symbole à été transmis. Cette décision sera faite "à la majorité" : la valeur du symbole apparaissant le plus de fois au sein des $J+1$ estimations sera retenue ($\tilde{I}$).

L'équation qui en découle est la suivante :

$$\tilde{I}^k = maj\{\hat{I}^k,\ \hat{P}_2^{k-g_0} + \sum_{i=1}^{J-1} I^{k-g_0+g_i},\ \hat{P}_2^{k-g_1} + \sum_{\substack{i=0 \\ i\neq 1}}^{J-1} I^{k-g_1+g_i},\ \hat{P}_2^{k-g_2} + \sum_{\substack{i=0 \\ i\neq 2}}^{J-1} I^{k-g_2+g_i}\}. \tag{3.3}$$

L'opérateur "*maj*" dénote tout simplement la recherche du symbole en majorité sur les équations contenues.
Dans cet exemple, les indices de parité j, $j+2$, $j+5$ sont utilisés respectivement avec les séquences d'information : $\{j-2, j-5\}$, $\{j+2, j-3\}$, $\{j+5, j+3\}$. On remarque que le code est bien simplement orthogonal car l'intersection des trois ensembles est vide.

$$\{j-2, j-5\} \cap \{j+2, j-3\} \cap \{j+5, j+3\} = \{\emptyset\} \tag{3.4}$$

Lors du décodage, afin de garantir un décodage itératif orthogonal, les symboles reçus utilisés au sein d'une itération doivent être distincts de ceux utilisés à l'itération précédente.

Observons ce qui se passe lors du fonctionnement itératif (on conserve l'exemple précédemment utilisé) :

- la première itération conduit à l'obtention de l'ensemble de différences simples : $\{j \pm (g_0 - g_1),\ j \pm (g_0 - g_2),\ j \pm (g_1 - g_2)\}$ (ou de manière équivalente $\{j \pm (g_i - g_k)\}$ avec i, k = 0, 1, 2 et $i \neq k$). Dès lors que l'on a $\{g_i - g_k\}$

distincts pour $i \neq k$, cet ensemble est orthogonal.

- la deuxième itération produit l'ensemble $\{j \pm [(g_i - g_k) - (g_l - g_m)]\}$. Les valeurs de nos générateurs choisis nous assurent que cet ensemble ne contient que des éléments distincts.

Revenons sur ce dernier point, la deuxième itération produit des éléments de la forme : $(g_i - g_k) - (g_l - g_m)$. Les conditions de restriction imposées sont les suivantes : $i \neq k$, $l \neq m$ (cas de l'orthogonalité simple (cf première itération)) et $k \neq l$ (équations de parité ne pouvant être réutilisées par elles même).

F. Gagnon et D. Haccoun ont établi grâce à diverses simulations qu'une condition d'orthogonalité au niveau de la seconde itération était suffisante pour aboutir à de très bonnes performances. Au-delà, il est certes possible d'augmenter le niveau des résultats, mais la complexité s'en ressent fortement et il est plus difficile et plus long de générer les bons codes.

3.3.2.4 Avantages

Une telle structure possède de nombreux avantages dont nous présentons un bref résumé ci dessous :

- architecture matérielle très simple,
- performances proches de la "qualité turbo",
- délai de traitement considérablement réduit par rapport aux codes turbo traditionnels,
- architecture facilement parallélisable et donc adaptée à traiter de forts débits de données.

Ces avantages sont sans aucun doute conséquents. Le seul problème réside dans la génération des codes utilisés.

3.4 Formalisme mathématique des codes doublement orthogonaux

3.4.1 Définition

Un code doublement orthogonal est tel que les différents observables mis en jeu au sein du système lors de la deuxième itération sont indépendants entre eux.

3.4.2 Mise en équations

D'un point de vue mathématique, la condition précédemment énoncée s'exprime sous la forme de trois contraintes à satisfaire pour les générateurs :

a) Différences simples distinctes entre elles

$$\begin{gathered} \forall(i,j)\ i > j,\ \forall(k,l)\ k > l \text{ et } (k,l) \neq (i,j) \\ g_i - g_j \neq g_k - g_l. \end{gathered} \tag{3.5}$$

b) Différences doubles distinctes des différences simples

$$\begin{gathered} \forall(i,j,k,l),\ i > j,\ i \geq l,\ j \geq k,\ l > k,\ j \neq l\ ;\ \forall(m,n)\ m > n \\ (g_i - g_j) - (g_k - g_l) \neq g_m - g_n. \end{gathered} \tag{3.6}$$

c) Différences doubles distinctes entre elles

$$\begin{gathered} \forall(i,j,k,l),\ i > j,\ i \geq l,\ j \geq k,\ l > k,\ j \neq l \\ \forall(i',j',k',l'),\ i' > j',\ i' \geq l',\ j' \geq k',\ l' > k',\ j' \neq l' \\ (i,j,k,l) \neq (i',j',k',l') \\ (g_i - g_j) - (g_k - g_l) \neq (g_{i'} - g_{j'}) - (g_{k'} - g_{l'}). \end{gathered} \tag{3.7}$$

En étudiant ces relations, on s'aperçoit qu'il est possible de les simplifier en éliminant les redondances.

Tout d'abord il est clair que l'on peut considérer, sans induire de restrictions, des ensembles de la forme $\{0, g_1, ..., g_{J-1}\}$. En effet, si l'ensemble $\{a_0, a_1, ..., a_{J-1}\}$ est doublement orthogonal, il en sera de même vu la forme des équations (3.5, 3.6 et 3.7) de l'ensemble : $\{a_0 - a_0, a_1 - a_0, ..., a_{J-1} - a_0\}$.

On montre à l'Annexe I que pour un ensemble de plus de quatre éléments, la condition 3.7 implique les deux autres (3.5 et 3.6).

La condition de double orthogonalité pour l'ensemble $\{0, g_1, ..., g_{J-1}\}$ se résume alors à l'obtention de :

$$\left\{\begin{array}{l} k > l \\ k \geq m \\ m > n \\ l \geq n \\ l \neq m \end{array}\right. \quad \left\{\begin{array}{l} k' > l' \\ k' \geq m' \\ m' > n' \\ l' \geq n' \\ l \neq m \end{array}\right. \quad (k, l, m, n) \neq (k', l', m', n') \qquad (3.8)$$

$$g_k - g_l + g_m - g_n \neq g_{k'} - g_{l'} + g_{m'} - g_{n'}.$$

3.4.2.1 Propriétés des ensembles d'entiers doublement orthogonaux

a) Origine

Ce point soulignant le fait que l'on peut toujours considérer un ensemble commençant par 0 a déjà été évoqué (3.4.2).

b) Existence de deux ensembles

Les ensembles trouvés vont toujours par paires. En effet si $\{0, g_1, ..., g_{J-1}\}$ est doublement orthogonal alors $\{g_{J-1} - g_{J-1}, g_{J-1} - g_{J-2}, ..., g_{J-1} - 0\}$ est également doublement orthogonal avec le même terme maximal g_{J-1}. La preuve de ceci est immédiate puisque si les combinaisons $g_i - g_j + g_k - g_l$ sont distinctes, il en sera de même pour $-g_i + g_j - g_k + g_l$, ce qui équivaut à $(g_{J-1} - g_i) - (g_{J-1} - g_j) + (g_{J-1} - g_k) - (g_{J-1} - g_l)$.

c) Équivalence différences distinctes - sommes distinctes

On montre à l'Annexe II que la condition $g_i - g_j + g_k - g_l$ distincts équivaut à $g_i + g_j + g_k + g_l$ distincts.

D'autres propriétés notamment vis à vis des opérations de multiplication et de modulo apparaîtront au cours de notre étude.

3.4.2.2 Différenciation ensembles complets / sans les négatifs

En réalité la condition exposée jusqu'ici est relativement spécifique et on appellera le type d'ensemble vérifiant celle-ci, des "ensembles doublement orthogonaux sans les négatifs".

L'appellation utilisée est claire dès lors que l'on étudie attentivement les restrictions imposées aux indices de (3.8). En effet, on a entre autres: $k > l$, $m > n$. Ceci fait en sorte que les différences simples $g_k - g_l$ et $g_m - g_n$ demeurent toujours positives.

Un autre cas a été envisagé: celui des "ensembles complets" au sein desquels l'on autorise les différences simples $g_k - g_l$ et $g_m - g_n$ à devenir négatives. Là encore

la condition de double orthogonalité qui implique trois conditions initialement se simplifie à une seule qui est la suivante:

$$\begin{cases} k \geq m \\ l \geq n \\ l \neq m \\ k \neq n \\ l \neq m \\ n \neq m \end{cases} \quad \begin{cases} k' \geq m' \\ l' \geq n' \\ l' \neq m' \\ k' \neq n' \\ l' \neq m' \\ n' \neq m' \end{cases} \quad (k, l, m, n) \neq (k', l', m', n') \tag{3.9}$$

$$g_k - g_l + g_m - g_n \neq g_{k'} - g_{l'} + g_{m'} - g_{n'}$$

$$ou$$

$$(g_k - g_l) - (g_n - g_m) \neq (g_{k'} - g_{l'}) - (g_{n'} - g_{m'}).$$

Quel intérêt y-a-t-il à considérer préférentiellement un type d'ensemble plutôt qu'un autre?

Les simulations réalisées indiquent que les ensembles complets permettent d'atteindre de meilleures performances. On peut énoncer une hypothèse pour tenter d'expliquer ce point. Autoriser les différences négatives consiste à orthogonaliser au sein de notre processus itératif les sorties d'une itération avec la séquence d'information utilisée à l'itération suivante. En procédant de la sorte, il est clair que les résultats obtenus seront meilleurs puisque l'on réduit par rapport aux ensembles avec les négatifs la corrélation entre les signaux considérés. En contrepartie, il sera plus difficile d'obtenir ce type d'ensembles. Ceux-ci sont en effet "moins nombreux" que les ensembles sans les négatifs qui les contiennent.

Dénombrement

Il est intéressant, et nous le réutiliserons par la suite (4.3.2), de dénombrer le nombre de différences simples et doubles que génère chacun des deux ensembles contenant J éléments.

- Ensembles sans les négatifs

Nombre de différences simples (N_s) :

C'est le nombre de combinaisons possibles de 2 éléments distincts parmi J soit

$$N_s = \begin{pmatrix} 2 \\ J \end{pmatrix} = \tfrac{1}{2}J(J-1)$$

Nombre de différences doubles (N_d) :

Le calcul est un peu plus compliqué puisqu'il faut prendre en compte la position relative des indices :

$$\begin{cases} k > l \\ m > n \\ m \neq l \quad g_k - g_l + g_m - g_n \\ l \geq n \\ k \geq m \end{cases} \tag{3.10}$$

Une façon d'aboutir au résultat est d'effectuer le raisonnement suivant :

1 On choisit k

De par $k > l$, k ne peut pas être égal à 0. C'est la seule restriction concernant

cette valeur. On a donc $\sum_{k=1}^{J-1}$ choix pour k.

2 On choisit l

Comme $k > l$, l est strictement inférieur à k. C'est la seule restriction qui s'applique ici soit $\sum_{l=1}^{k-1}$ choix pour l (k fixé).

3 On choisit m

On distingue deux cas (on sait que $m \neq l$) :

+ Si $m < l$

D'après $m > n$ on sait que l'on a forcément $m > 0$. Par hypothèse, $m < l$ ce qui impose qu'il y a : $\sum_{m=1}^{l-1}$ possibilités pour m (k et l fixés).

4 On choisit n

Avec $m > n$, on obtient qu'il y a $\sum_{n=0}^{m-1}$ choix pour n (m, k et l fixés).

+ Si $m > l$

D'après $k \geq m$ et avec l'hypothèse $m > l$ il y a : $\sum_{m=1}^{l-1}$ possibilités pour m (k et l fixés).

4 On choisit n

On doit satisfaire aux deux conditions $m > n$ et $l \geq n$. Comme $m > l$, il ne reste que la contrainte $l \geq n$ ce qui donne : $\sum_{n=0}^{l}$ choix possibles pour n (m, k et l fixés).

En regroupant toutes les possibilités, on obtient :

$$N_d = \sum_{k=1}^{J-1} \sum_{l=0}^{k-1} (\sum_{m=1}^{l-1} \sum_{n=0}^{m-1} + \sum_{m=l+1}^{k} \sum_{n=0}^{l})$$
$$N_d = \sum_{k=1}^{J-1} (\sum_{l=2}^{k-1} \sum_{m=1}^{l-1} \sum_{n=0}^{m-1} + \sum_{l=0}^{k-1} \sum_{m=l+1}^{k} \sum_{n=0}^{l})$$
$$N_d = \sum_{k=1}^{J-1} (\sum_{l=2}^{k-1} \sum_{m=1}^{l-1} m + \sum_{l=0}^{k-1} \sum_{m=l+1}^{k} (l+1))$$
$$N_d = \sum_{k=1}^{J-1} (\sum_{l=2}^{k-1} \tfrac{1}{2} l(l-1) + \sum_{l=0}^{k-1} (l+1)(k-l))$$
$$N_d = \sum_{k=1}^{J-1} (\tfrac{1}{6}k^3 - \tfrac{1}{2}k^2 + \tfrac{1}{3}k + \tfrac{1}{2}k^2 + \tfrac{1}{6}k^3 + \tfrac{1}{3}k)$$
$$N_d = \sum_{k=1}^{J-1} \tfrac{1}{3}(k^3 + 2k)$$

$$N_d = \frac{1}{12} J(J-1)(J^2 - J + 4)$$

En utilisant successivement :

$$\sum_{i=0}^{N} i = \frac{1}{2} N(N+1)$$
$$\sum_{i=0}^{N} i^2 = \frac{1}{6} N(N+1)(2N+1)$$
$$\sum_{i=0}^{N} i^3 = \frac{1}{4} N^2(N+1)^2$$

Ainsi on a :

$$N_s = \frac{1}{2} J(J-1)$$
$$N_d = \frac{1}{12} J(J-1)(J^2 - J + 4)$$
$$N_s + N_d = \frac{1}{12} J(J-1)(J^2 - J + 10)$$

- Ensembles complets

Nombre de différences simples (N_s) :

C'est le nombre d'arrangements possibles de 2 éléments distincts parmi J soit

$$N_s = A_J^2 = J(J-1)$$

Nombre de différences doubles (N_d) :

Là encore le calcul est un peu plus compliqué puisqu'il faut prendre en compte la position relative des indices :

$$\begin{cases} k \geq m \\ i \geq l \\ k \neq i \\ m \neq l \\ i \neq m \\ k \neq l \end{cases} \quad g_k - g_i + g_m - g_l \qquad (3.11)$$

1 On choisit k

Cette grandeur pouvant prendre n'importe quelle valeur, on a $\sum_{k=1}^{J}$ choix possibles.

2 On choisit i

Comme on a $i \neq k$, on est confronté aux deux cas suivants $i > k$ ou $i < k$.

+ Si $i > k$

On a alors $\sum_{i=k+1}^{J}$ possibilités pour i.

3 On fixe m ($m \neq k$)

Avec les conditions $k \geq m$ et $m \neq i$ et sachant que $i > k$ on aboutit à $\sum_{m=1}^{k-1}$ possibilités pour m.

4 On fixe l

Les contraintes nous imposent $i \geq l$, $l \neq m$, $l \neq k$ comme $i > k$ et $m \leq k$ on aboutit à $\sum_{l=1}^{i-2}$ possibilités pour m.

On a donc pour ce chemin global $\sum_{k=1}^{J} \sum_{i=k+1}^{J} \sum_{m=1}^{k-1} \sum_{l=1}^{i-2}$

3 Si $m = k$

On n'a naturellement qu'un unique choix pour m.

4 On fixe l

Les contraintes demeurent inchangées à savoir : $i \geq l$, $l \neq m$, $l \neq k$. Comme $m = k$ les deux dernières inégalités n'en font qu'une. Soit $i \geq l$, $l \neq m$. Sachant que $i > k$ on obtient $\sum_{l=1}^{i-1}$ possibilités pour l.

Le trajet global dans ce cas là conduit à $\sum_{k=1}^{J} \sum_{i=k+1}^{J} .1. \sum_{l=1}^{i-1}$ possibilités.

+ Si $i < k$

On a nécessairement $\sum_{i=1}^{k-1}$ choix possibles pour i.

3 On fixe m

On doit respecter les contraintes $k \geq m$ et $m \neq i$ sachant que $i < k$. On distingue les deux cas où $m > i$ et $m < i$ pour étudier de manière

convenable le cas de l.

++ Si $m < i$

Comme $i < k$ on a forcément $m \geq k$. Ceci laisse $\sum_{m=1}^{i-1}$ possibilités pour m.

4 On fixe l

l doit vérifier $i \geq l$, $l \neq m$, $l \neq k$ comme $i < k$ et $i \geq l$, la condition $l \neq k$ est toujours vraie. Il ne reste donc plus que $i \geq l$ et $l \neq m$. Ceci laisse $\sum_{l=1}^{i-1}$ possibilités pour l.

Ce parcours induit donc $\sum_{k=1}^{J} \sum_{i=1}^{k-1} \sum_{m=1}^{i-1} \sum_{l=1}^{i-1}$ choix.

++ Si $m > i$

il faut que m vérifie $m \geq k$ ce qui impose pour choisir m: $\sum_{m=i+1}^{k}$ possibilités.

4 On fixe l

l doit être choisi tel que $i \geq l$, $l \neq m$ et $l \neq k$ avec $i < k$ et $i \geq l$ la condition $l \neq k$ est toujours vérifiée. Avec $i \geq l$ et $m > i$ on aura toujours $l \neq m$. La seule restriction concernant l est donc $i \geq l$. Ceci laisse $\sum_{l=1}^{i}$ choix pour l.

Ce trajet conduit donc à $\sum_{k=1}^{J} \sum_{i=1}^{k-1} \sum_{m=i+1}^{k} \sum_{l=1}^{i}$ possibilités.

En regroupant tous les cas on obtient l'expression:

$$N_d =$$

$$\sum_{k=1}^{J}(\sum_{i=k+1}^{J} \sum_{m=1}^{k-1} \sum_{l=1}^{i-2} + \sum_{i=k+1}^{J} \sum_{l=1}^{i-1} + \sum_{i=1}^{k-1}(\sum_{m=1}^{i-1} \sum_{l=1}^{i-1} + \sum_{m=i+1}^{k} \sum_{l=1}^{i}))$$

$$N_d = \sum_{k=1}^{J}(\underbrace{\sum_{i=k+1}^{J} \sum_{m=1}^{k-1} \sum_{l=1}^{i-2}}_{\frac{1}{2}(k-1)(J-3+k)(J-k)} + \underbrace{\sum_{i=k+1}^{J} \sum_{l=1}^{i-1}}_{\frac{1}{2}(J-k)(J-1+k)} + \underbrace{\sum_{i=1}^{k-1}(\underbrace{\sum_{m=1}^{i-1} \sum_{l=1}^{i-1}}_{(i-1)^2} + \underbrace{\sum_{m=i+1}^{k} \sum_{l=1}^{j}}_{(k-i)i})}_{\frac{1}{2}(k-1)(k^2-2k+2)})$$

$$N_d = \sum_{k=1}^{J}(\tfrac{1}{2}(J-1)(kJ+2-2k))$$

$$N_d = \tfrac{1}{4}J(J-1)(J^2-J+2)$$

Ainsi on a :

$$N_s = J(J-1)$$
$$N_d = \tfrac{1}{4}J(J-1)(J^2-J+2)$$
$$N_s + N_d = \tfrac{1}{4}J(J-1)(J^2-J+6)$$

3.4.2.3 Différenciation orthogonalité sens strict / sens large

Il est à noter que la condition de double orthogonalité présentée jusqu'à présent n'était qu'un formalisme mathématique qui a été utilisé tout au long de notre étude.

Si l'on en revient à une signification plus physique, il n'est pas souhaitable d'imposer les restrictions $k \geq m$ et $l \geq n$ au sein de l'équation 3.9. Ceci impliquerait en effet en pratique une suppression d'un certain nombre d'équations de parité pouvant conduire à une propagation d'erreur.

La condition de double orthogonalité ainsi considérée (par exemple pour des ensembles complets) fait apparaître dans certains cas des permutations inévitables puisque $g_i - g_j + g_k - g_l = g_i - g_l + g_k - g_j$. Ceci implique qu'on ne peut pas parler de double orthogonalité au sens strict du terme et on dénommera orthogonalité au sens large cette propriété.

On verra cependant au 5.1 qu'il est possible d'obtenir une double orthogonalité au sens strict en adoptant une architecture matérielle particulière pour le codeur.

CHAPITRE 4

CODES DOUBLEMENT ORTHOGONAUX DE TAUX 1/2 AU SENS LARGE

4.1 Introduction

Ce chapitre porte spécifiquement sur la génération de codes de taux $\frac{1}{2}$ doublement orthogonaux (d.o.) au sens large. Les algorithmes de génération utilisés y sont présentés ainsi que les différentes techniques de réduction. La combinaison de ces procédures a permis d'aboutir à l'obtention de générateurs aux longueurs réduites.

4.2 Génération

4.2.1 Géométrie projective

4.2.1.1 Description

La construction et les règles d'orthogonalisation de la plupart des codes cycliques sont basées sur la structure de géométries finies comme par exemple la géométrie euclidienne et la géométrie projective. Ces codes appelés codes à géométrie finie ont été initialement étudiés par Rudolph [47].

La géométrie projective se construit à partir des éléments d'un corps de Galois [4]. Un corps de Galois est l'appellation donnée (du nom de son découvreur) aux corps possédant un nombre d'éléments fini. Le nombre d'éléments (z) de ce corps détermine sa notation sous la forme GF(p^d) où p est un nombre premier et où $z = p^d$. L'entier p est appelé le nombre caractéristique du corps, c'est le plus petit élément tel que l'addition de p fois l'élément unité du corps donne pour résultat l'élément

nul. On nomme élément primitif α, tout élément d'ordre $p^d - 1$ (ie: $\alpha^{p^d-1} = \alpha$). Par définition, un élément primitif α d'un corps de Galois GF($p^{s(m+1)}$) est une des racines d'un polynôme primitif d'ordre $m+1$ à valeurs dans GF(p^s). Notons $g(x)$ ce polynôme, il est défini comme étant irréductible sur le corps considéré et de sorte à ce que le plus petit entier n tel que $g(x)$ divise $x^n - 1$ soit égal à $p^{s(m+1)}$. Ce polynôme ne peut être le produit de deux polynômes de moindre degré du corps (on peut voir dans cette dernière définition un rapprochement avec les nombres premiers).

On considère alors, pour fixer les idées, des corps de Galois du type GF($2^{s(m+1)}$). Ceux-ci contiennent GF(2^s) comme sous corps. On note α un élément primitif de GF($2^{s(m+1)}$). Voyons sur un exemple appliqué au codage la construction du corps contenant les 16 quatruplets binaires (de 0000 à 1111) soit GF(16)=GF(2^4). On affecte à chaque élément un polynôme spécifique: 0000 $\rightarrow$ 0, 1000 $\rightarrow$ 1, 0100 $\rightarrow$ x, 1100 $\rightarrow 1+x$, 0010 $\rightarrow x^2$, ..., 1111 $\rightarrow 1+x+x^2+x^3$. Une brève recherche montre que le polynôme x^4+x+1 est primitif sur GF(16). Dénotons par α l'élément primitif racine de ce polynôme. Le tableau récapitulatif des éléments de cet ensemble est alors:

Tableau 4.1 - *Construction de GF(16)*

élément	polynôme	puissance	élément	polynôme	puissance
0000	0	0	1101	$1+\alpha+\alpha^3$	α^7
1000	1	1	1010	$1+\alpha^2$	α^8
0100	α	α	0101	$\alpha+\alpha^3$	α^9
0010	α^2	α^2	1110	$1+\alpha+\alpha^2$	α^{10}
0001	α^3	α^3	0111	$\alpha+\alpha^2+\alpha^3$	α^{11}
1100	α^4	α^4	1111	$1+\alpha+\alpha^2+\alpha^3$	α^{12}
0110	α^5	α^5	1011	$1+\alpha^2+\alpha^3$	α^{13}
0011	α^6	α^6	1001	$1+\alpha^{13}$	α^{14}

Les puissances de α: α^0, α^1, ..., $\alpha^{2^{s(m+1)}-2}$ forment alors les éléments non-nuls de GF($2^{s(m+1)}$). Définissons n entier tel que: $n = \frac{2^{(m+1)s}-1}{2^s-1} = 2^{ms} + 2^{(m-1)s} +$

$... + 2^s + 1$. En posant $\beta = \alpha^n$, il est clair que l'ordre de β est $2^s - 1$. Les 2^s éléments $\{0, 1, \beta, \beta^2, ..., \beta^{2^s-2}\}$ constituent le corps de Galois GF(2^s). Notons $\Gamma = \{\alpha^0, \alpha^1, ..., \alpha^{n-1}\}$ (les n premières puissances de α). Partitionnons les éléments non-zéro de GF($2^{(m+1)s}$) en n sous-ensembles disjoints:

$$\{\alpha^0, \beta.\alpha^0, ..., \beta^{2^s-2}.\alpha^0\},$$
$$\{\alpha^1, \beta.\alpha^1, ..., \beta^{2^s-2}.\alpha^1\},$$
$$\vdots$$
$$\{\alpha^{n-1}, \beta.\alpha^{n-1}, ..., \beta^{2^s-2}.\alpha^{n-1}\}.$$

Proposition:

Chaque ensemble a $2^s - 1$ éléments multiples du premier élément. Aucun élément de ces ensembles ne peut être un produit d'un élément de GF(2^s) et d'un élément d'un ensemble différent.

Preuve:

Montrons tout d'abord par l'absurde qu'aucun élément α^i de Γ ne peut être un produit d'un élément de GF(2^s) et d'un autre élément α^j de Γ. Si $\alpha^i = \eta.\alpha^j$ avec η appartenant à GF(2^s), on a alors comme $\eta^{2^s-1} = 1$, $\alpha^{(i-j)(2^s-1)} = 1$. Avec $(i - j).(2^s - 1) < 2^{(m+1)s} - 1$ cela signifierait que l'ordre de α serait inférieur à $2^{(m+1)s} - 1$ ce qui est impossible. Ceci établit, il est évident maintenant que si l'on a $\beta^i.\alpha^k = \beta^m.\alpha^j.\beta^p$ (expression du produit d'un élément des ensembles définis précédemment et d'un élément de GF(2^s)) on aboutit à l'équation $\alpha^k = \alpha^j.\beta^{m+p-i}$ ce qui, on l'a vu auparavant, est impossible. La proposition est ainsi démontrée.

La géométrie projective comme son nom l'indique utilise le concept de projection. En fait chaque ensemble précédemment explicité est représenté par son premier élément (ie: $(\alpha^i) = \{\alpha^i, \beta.\alpha^i, ..., \beta^{2^s-2}.\alpha^i\}$). Ainsi pour tout α^j dans GF($2^{s(m+1)}$))

si $\alpha^k = \beta^i.\alpha^j$ avec $0 \leq j < n$ alors α^j est représenté par (α^k). On dit que les éléments (α^0), (α^1), ..., (α^{n-1}) forment une géométrie projective à n dimensions sur GF(2^s) et sont appelés des points.
Remarque: $\{\alpha^i, \beta\alpha^i, ..., \beta^{2^s-2}\alpha^i\}$ représente le même point (projection)

Lignes: soient (α^i) et (α^j) deux points distincts dans (PG(m, 2^s)) (PG(m, 2^s) sous-entend la structure de géométrie projective définie sur le corps GF($2^{s(m+1)}$)). La ligne passant par ces deux points consiste en les points $(\eta_1.\alpha i + \eta_2.\alpha j)$. S'il y a $(2^s)^2 - 1$ choix pour η_1 et η_2 à partir de GF(2^s) (en excluant $\eta_1 = \eta_2 = 0$) il y a toujours $2^s - 1$ choix de η_1 et η_2 qui résultent en l'obtention de la même ligne $\{\eta_1.\alpha i + \eta_2.\alpha j,\ \eta_1.\beta.\alpha i + \eta_2.\beta.\alpha j,\ ...,\ \eta_1.\beta^{2^s-2}.\alpha i + \eta_2.\beta^{2^s-2}.\alpha j\}$. Une ligne dans PG($m$, 2^s) "contient" donc $\frac{(2^s)^2-1}{2^s-1} = 2^s + 1$ points.

μ-plans: soient (α^{t_1}), (α^{t_2}), ..., $(\alpha^{t_{\mu+1}})$ des points linéairement indépendants. Un μ-plan dans PG(m, 2^s) est défini par les points $(\eta_1\alpha^{t_1} + \eta_2\alpha^{t_2} + ... + \eta_{\mu+1}\alpha^{t_{\mu+1}})$ où les η_i appartiennent à GF(2^s) et tels qu'ils ne soient pas tous nuls. En tenant compte du fait qu'il y a toujours $2^s - 1$ choix de η_1, ..., $\eta_{\mu+1}$ qui aboutissent au même point dans PG(m, 2^s) il y a $\frac{2^{(\mu+1)s}-1}{2^s-1} = 1+2^s+...+2^{\mu s}$ points dans un μ-plan de PG(m, 2^s).

Cette description n'avait pas pour vocation d'expliciter en détails la matière que représente la géométrie projective. Elle est néanmoins suffisante pour la compréhension de l'utilisation qui en a été faite durant notre étude.

4.2.1.2 Principe utilisé

Le principe utilisé pour générer des codes M-orthogonaux à partir de la géométrie projective est présenté ci-après. Afin d'alléger les notations, on traitera en détails le cas de codes simplement orthogonaux. La généralisation à des ordres d'orthogonalité supérieurs est immédiate.

Considérons un corps de Galois GF($p^{s(m+1)}$). Les paramètres m, s et p sont choisis de sorte à ce que le nombre de générateurs du code considéré J soit égal à $p^s + 1$ avec p premier et s entier. Si une telle égalité n'est pas possible on prend les valeurs de p et de s qui se rapprochent le plus de J par valeurs supérieures. Le paramètre m renvoie au niveau d'orthogonalité souhaité. Ainsi on a $m = 2$ pour des codes simplement orthogonaux, $m = 4$ pour des codes doublement orthogonaux et ainsi de suite. Notons α un élément primitif de ce corps GF($p^{s(m+1)}$). Par définition, un élément primitif d'un corps de Galois α est une des racines d'un polynôme primitif d'ordre $m + 1$ à valeurs dans GF(p^s).

Associons à ce corps de Galois une structure de géométrie projective PG(m, p^s). On sait d'après la section précédente que l'ensemble ainsi généré contient $n = \frac{p^{s(m+1)}-1}{p^s-1}$ points. En reprenant les notations précédentes on note $\beta = \alpha^n$. L'élément β est un élément primitif du corps de Galois GF(p^s). On génère alors simplement une ligne dans PG($m = 2$, p^s).

Proposition :

Les puissances de α prises sur une ligne particulière de PG(m, p^s) forment un ensemble de valeurs vérifiant la propriété de simple orthogonalité.

Preuve :

(α) étant d'ordre n il est clair que l'ensemble de ses puissances est défini modulo n. On restreindra la forme de la ligne précédemment choisie au type $\eta_1 + \eta_2.\alpha$ où η_1 et η_2 sont des éléments de GF(p^s). On notera par la suite $g_0, g_1, ..., g_{J-1}$ les générateurs recherchés. Le premier "point" sur la ligne considérée est : $(\alpha^0) = (\alpha^{g_0}) = (1)$, le deuxième est $(\alpha^1) = (\alpha^{g_1}) = (\alpha)$ et les suivants sont du type $(\alpha^{g_i}) = (1 + \beta^{i-2}.\alpha)$. Prenons par exemple $(\alpha^{g_i}) = (1 + \beta^{i-2}.\alpha)$ et $(\alpha^{g_j}) = (1 + \beta^{j-2}.\alpha)$, nommons pour

clarifier la notation $\mu_i = \beta^{i-2}$ et $\mu_j = \beta^{j-2}$. Le produit de (α^{g_i}) avec (α^{g_j}) donne :

$$\alpha^{g_i}.\alpha^{g_j} = (1 + \mu_i.\alpha).(1 + \mu_j.\alpha) = 1 + (\mu_i + \mu_j).\alpha + \mu_i.\mu_j.\alpha^2.$$

Soit $k_{i,j}$ tel que :

$$\alpha^{k_{i,j}} = (1 + \mu_i.\alpha).(1 + \mu_j.\alpha).$$

On sait que les racines de tous les polynômes générés par le produit de deux points sont distinctes (4.2.1.1), et que de plus, le produit de deux points aboutit à un polynôme de degré deux au maximum ce qui est inférieur au degré du polynôme primitif (ordre $m + 1 = 3$) et évite donc la nécessité d'appliquer l'opération de modulo. Ainsi les nombres $k_{i,j}$ peuvent être utilisés pour représenter des éléments distincts dans GF($p^{s(m+1)}$) avec $k_{i,j} = g_i + g_j$. Comme aucun polynôme n'est le multiple d'un autre avec les scalaires pris dans GF(p^s), la projection des éléments sur PG(2, p^s) conduit à des résultats distincts. Il est à noter que l'on génère ainsi des ensembles complets qui sont topologiquement inclus dans ceux "sans les négatifs" (3.4.2.2).

La construction présentée ci dessus peut être facilement généralisée aux valeurs de m supérieures. Il suffit pour cela de générer une ligne dans PG(m, p^s). Il convient cependant d'être prudent lors de l'équivalence différences - sommes que l'on n'a pas démontré dans le cas général pour $m > 4$ (Annexe II).

4.2.1.3 Mise en oeuvre

La mise en oeuvre algorithmique de la procédure précédente a été effectuée comme suit :

- saisie de la valeur de J par l'utilisateur,
- détermination par une simple procédure testant les différents cas les valeurs de p et de s correspondantes au J en entrée (m est fixé à 4 pour des codes doublement orthogonaux),

- génération d'un polynôme primitif de $GF(p^{s(m+1)})$ par une procédure écrite en langage C (principe: division et opération de modulo sur les polynômes),
- recherche de l'ensemble de nombres doublement orthogonaux obtenu par une procédure utilisant le logiciel *Maple*© (principe: on estime modulo le polynôme primitif les grandeurs $\alpha^{g_i} - (1 + \alpha.\alpha^{n.j})$. Si le degré du polynôme obtenu est égal ou inférieur à zéro le générateur g_i convient).

4.2.1.4 Résultats obtenus

Les résultats présentés dans le tableau 4.2 ont été établis suivant la méthode explicitée dans la section précédente. Ils regroupent les meilleures valeurs obtenues pour des codes d.o. au sens large de taux $\frac{1}{2}$ dont le nombre de générateurs (J) varie de 2 à 20. Le paramètre modulo apparaissant dans le tableau correspond au "n", ordre de l'élément primitif auparavant décrit.

La procédure de génération s'avère relativement lente, surtout pour les valeurs de J élevées. Sans nul doute l'utilisation d'un logiciel de calcul tel que *Maple*© ralentit de beaucoup la simulation. Un algorithme écrit en langage C eût permis d'améliorer le temps nécessaire au calcul, cependant ce serait se priver de nombreuses fonctionnalités incluses dans les logiciels de calcul formel notamment au niveau des opérations sur les corps de Galois bien utiles ici.

Une première analyse fait apparaître une longueur de ces codes très élevée. Une réduction semble donc obligatoire et sera présentée par la suite. Les cas $J \geq 15$ ne seront plus étudiés par la suite. Ceux-ci sont en effet peu intéressants asymptotiquement car ils conduisent à des longueurs de code extrêmement élevées inexploitables en pratique, sans pour autant fournir des performances d'erreur proportionnelles à ces longueurs.

Tableau 4.2 – *Générateurs obtenus par géométrie projective pour des codes de taux* $\frac{1}{2}$ *à J éléments*

J	générateurs	paramètres		
		p	s	modulo
2	{0, 1}	2	1	31
3	{0, 1, 18}	2	1	31
4	{0, 1, 5, 69}	3	1	121
5	{0,1, 32, 77, 242}	2	2	341
6	{0, 1, 5, 25, 125, 625}	5	1	781
7	{0, 1, 5, 841, 867, 1176, 1237}	7	1	2801
8	{0, 1, 5, 841, 867, 1176, 1237, 2510}	7	1	2801
9	{0, 1, 15, 215, 1128, 1674, 1766, 3181, 3723}	2	3	4681
10	{0, 1, 963, 3317, 3673, 3933, 4490, 5945, 6315, 6965}	3	2	7381
11	{0, 1, 3729, 8536, 10148, 10865, 12749, 13031, 13603, 14093, 14236}	11	1	16105
12	{0, 1, 3729, 8536, 10148, 10865, 12749, 13031, 13603, 14093, 14236, 15367}	11	1	16105
13	{0, 1, 5, 2445, 3431, 4235, 5965, 8663, 15129, 16307, 16580, 17694, 18673}	13	1	30941
14	{0, 1, 5, 2445, 3431, 4235, 5965, 8663, 15129, 16307, 16580, 17694, 18673, 24791}	13	1	30941
15	{0, 1, 5, 7982, 15760, 16210, 22066, 23146, 24392, 38186, 44950, 50004, 56920, 66165, 76709}	17	1	88741
16	{0, 1, 5, 7982, 15760, 16210, 22066, 23146, 24392, 38186, 44950, 50004, 56920, 66165, 76709, 81999}	17	1	88741
17	{0, 1, 5, 7982, 15760, 16210, 22066, 23146, 24392, 38186, 44950, 50004, 56920, 66165, 76709, 81999, 82344}	17	1	88741
18	{0, 1, 5, 7982, 15760, 16210, 22066, 23146, 24392, 38186, 44950, 50004, 56920, 66165, 76709, 81999, 82344, 84431}	17	1	88741
19	{0, 1, 5, 10218, 10969, 17805, 24059, 32369, 39198, 41742, 56836, 57737, 58103, 61787, 74828, 77642, 98476, 104056, 112999}	19	1	137561
20	{0, 1, 5, 10218, 10969, 17805, 24059, 32369, 39198, 41742, 56836, 57737, 58103, 61787, 74828, 77642, 98476, 104056, 112999, 125172}	19	1	137561

4.2.2 Géneration pseudo-aléatoire

4.2.2.1 Présentation

L'utilisation de la géométrie projective ne donne pas entière satisfaction. Le temps de calcul est long et les ensembles obtenus possèdent une longueur excessive. Bien qu'il soit possible par diverses méthodes de réduire celle-ci, il peut être préférable d'appliquer ces techniques à un ensemble déjà plus réduit. Ceci justifie la recherche d'autres procédures de génération.

L'idée employée ici est simple et ne fait pas appel à des structures algébriques compliquées comme pouvait le faire la géométrie projective. Cette méthode emploie en réalité une procédure aléatoire qu'il est possible de modifier et d'affiner pour améliorer les résultats.

Le déroulement de cette technique peut être modélisée par la succession d'étapes suivantes :

1. On part d'un ensemble ne comprenant que le chiffre $\{0\}$ (cf propriétés d.o.), on initialise une variable entière "compteur" à 0.
2. On incrémente d'une unité la valeur contenue dans la variable "compteur".
3. On rajoute à l'ensemble en cours d'étude l'élément contenu dans "compteur".
4. On teste si ce nouvel ensemble vérifie la propriété de double orthogonalité.
5. Si le test échoue on retire l'élément de la variable "compteur" à notre ensemble et on réitère le processus à partir de l'étape $n°2$.
6. Sinon on effectue un tirage aléatoire. Si le résultat de ce tirage n'est pas validé on retire l'élément de la variable "compteur" à notre ensemble et on réitère le processus à partir de l'étape $n°2$.

7. Si le tirage aléatoire est accepté il ne reste plus qu'à recommencer à partir de l'étape $n°2$ en conservant l'ensemble obtenu par l'ajout de "compteur" (le nombre d'éléments augmente d'une unité). On prend soin de vérifier avant cela que le nombre d'élément requis J n'est pas atteint.

Cette procédure est répétée un certain nombre de fois pour ne conserver que le meilleur ensemble obtenu.

Cette structure se prête facilement à une représentation "du type graphcet". Celle-ci est présentée à la figure 4.1.

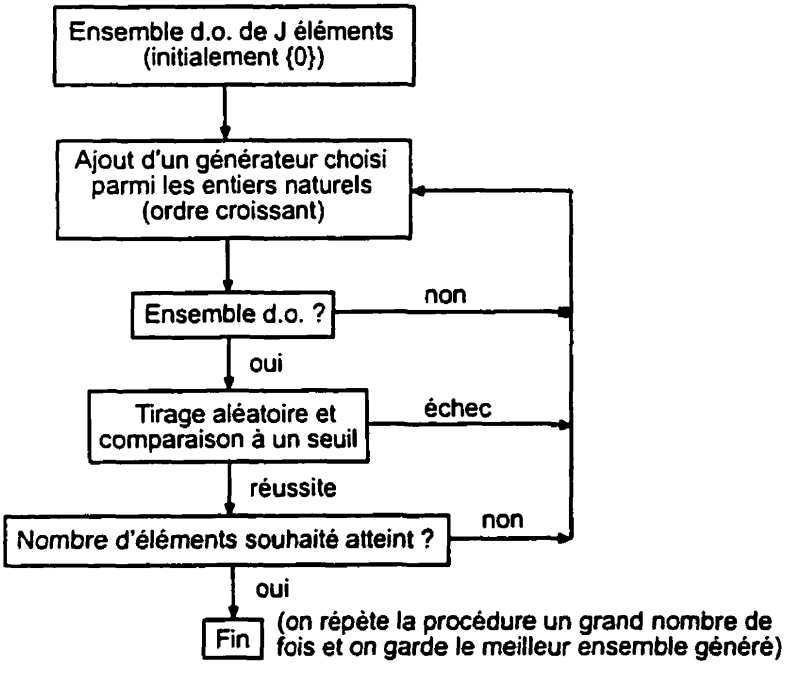

Figure 4.1 – *Processus de génération pseudo-aléatoire*

Le point le plus délicat de cette méthode est évidemment la détermination du tirage aléatoire à effectuer. Plusieurs solutions ont été envisagées et deux principales se basant sur l'utilisation d'un seuil ont été retenues.

a) Seuil fixe

La solution appréhendée ici est en fait celle d'un simple tirage aléatoire semblable à l'exemple classique d'un lancement de dés. On génère de manière aléatoire un nombre compris entre une valeur minimale et une valeur maximale et on fixe un seuil au dessous duquel le tirage est considéré comme "vérifié" (i.e.: on conserve l'élément considéré pour compléter notre ensemble d.o.).

Lors de l'implémentation il convient d'être vigilant eu égard à l'obtention d'un nombre réellement aléatoire. Les générateurs standards inclus dans les librairies de programmation sont bien souvent limités et font intervenir des paramètres de temps pour simuler le caractère aléatoire. Il n'est pas rare que ces configurations donnent les mêmes suites aléatoires lors de lancements consécutifs d'un même programme ou au bout d'un certain temps de simulation. Ce problème de génération de nombres aléatoires purs a été et est toujours l'objet de nombreuses recherches; il est possible d'en trouver de très performants sur Internet [51]. Plusieurs furent testés dans le cadre de cette étude. En pratique le critère retenu pour le tirage aléatoire fut le suivant:

. valeur minimale: 0, valeur maximale inférieure à 1,
. si le tirage produit une valeur inférieure à 0.2 le test est validé.

Le nombre 0.2 a été obtenu après expérimentations. Quelques exemples de ces simulations sont présentés dans le tableau 4.3 sur une moyenne de 20 essais lors de la génération d'un ensemble de 7 éléments:

Tableau 4.3 - *Évolution de la longueur moyenne en fonction de la valeur du seuil pour* $J = 7$

valeur	longueur moyenne obtenue
0.1	336
0.15	321
0.2	302
0.25	310
0.3	315
0.4	351

Dans ces conditions la génération s'avère rapide et produit de bons résultats. Cependant on s'aperçoit par rapport aux meilleures valeurs obtenues par la suite que plus le nombre d'éléments est grand et plus la longueur maximale s'éloigne de celle optimale trouvée après réduction. Quelques résultats sont présentés à la fin de cette partie en différenciant les ensembles complets des ensembles sans les négatifs.

Il est à noter que différentes variantes de cette technique ont été employées, notamment en faisant évoluer le paramètre aléatoire au cours d'une même génération. La meilleure obtenue est celle consistant à adopter un seuil linéaire explicitée ci-après.

b) Seuil linéaire

Cette méthode reprend les attributs de la technique précédente. Le seuil est défini sous la forme $a.x + b$ où x est le "numéro" de l'élément recherché et a et b sont deux paramètres tels que :

$$\begin{cases} 0.1 = a.1 + b \\ 1 = a.J + b \end{cases} \Rightarrow \begin{cases} a = \frac{0.9}{J-1} \\ b = 0.1J - 1 \end{cases} \tag{4.1}$$

On peut justifier ce choix de manière assez simple. Plus on se rapproche du nombre d'éléments maximum de notre ensemble (J) plus il semble intéressant de prendre le premier nombre trouvé. Pour passer de $J-1$ à J générateurs on obtiendra naturellement une longueur moins importante en prenant le premier qui convient. Lorsque l'on choisit les premiers éléments, il est possible de rejeter plus de choix sans que cela soit préjudiciable. Ceci se voit facilement ; par exemple pour le premier élément 1 convient forcément, si celui ci est rejeté, on sait que 2 conviendra aussi puis 3, 4, ... et donc que l'on obtiendra un premier élément en moyenne inférieur à 10 avec les valeurs de a et b retenues. Si on rejette un nombre aussi important d'éléments lorsqu'on recherche les derniers générateurs de notre ensemble, on est sûr d'obtenir une longueur excessivement élevée. En effet, les générateurs valides sont de plus en plus espacés au fur et à mesure de l'augmentation du nombre d'éléments trouvés.

En regardant de plus près les deux techniques retenues, celles-ci peuvent sembler contradictoires. En effet dans un cas (seuil fixe) on choisit un seuil faible pour les "derniers" générateurs et dans l'autre cas (seuil linéaire), la stratégie inverse est adoptée. Le caractère aléatoire de ces méthodes fait qu'en moyenne chacune des deux s'avère intéressante.

4.2.2.2 Résultats obtenus

- ensembles sans les négatifs

Tableau 4.4 - *Génération d'ensembles sans les négatifs par méthode pseudo-aléatoire*

J	générateurs
2	{0, 1}
3	{0, 1, 4}
4	{0, 3, 11, 12}
5	{0, 2, 23, 28, 34}
6	{0, 15, 31, 89, 94, 96}
7	{0, 6, 48, 53, 175, 183, 198}
8	{0, 1, 4, 40, 150, 323, 368, 379}
9	{0, 29, 75, 404, 864, 983, 1047, 1048, 1051}
10	{0, 36, 104, 474, 810, 1304, 1804, 1915, 1939, 1960}

- ensembles complets

Tableau 4.5 - *Génération d'ensembles complets par méthode pseudo-aléatoire*

J	générateurs
2	{0, 1}
3	{0, 2, 5}
4	{0, 4, 14, 15}
5	{0, 1, 37, 41, 51}
6	{0, 1, 6, 34, 76, 167}
7	{0, 1, 5, 23, 93, 197, 340}
8	{0, 49, 67, 173, 420, 606, 617, 620}
9	{0, 29, 75, 404, 864, 983, 1047, 1048, 1051}
10	{0, 36, 104, 474, 810, 1304, 1804, 1915, 1939, 1960}

Ces résultats seront interprétés par la suite à la section "comparaison des résultats".

4.2.3 Autres méthodes étudiées

a) Recherche exhaustive

C'est naturellement la méthode la plus simple et la plus précise pour générer des ensembles doublement orthogonaux de longueur minimale. Malheureusement une telle recherche est énormément coûteuse en terme de nombre de calculs à effectuer. Devant envisager tous les cas, elle fait appel à une structure d'imbrications qui sont "lourdes" à traiter. Ce fait limite l'utilisation de cette méthode à de faibles valeurs de J à moins de disposer d'un ordinateur très puissant. Nous n'avons pas pu dans notre cas pousser cette recherche exhaustive plus loin que la valeur $J = 5$. Dans les tableaux 4.6 et 4.7 figurent respectivement les résultats obtenus pour les ensembles complets et sans les négatifs.

- ensembles sans les négatifs

Tableau 4.6 – *Génération d'ensembles sans les négatifs par recherche exhaustive*

J	générateurs
2	{0, 1}
3	{0, 1, 4}
4	{0, 3, 11, 12}
5	{0, 1, 4, 24, 33}

- ensembles complets

Tableau 4.7 – *Génération d'ensembles complets par recherche exhaustive*

J	générateurs
2	{0, 1}
3	{0, 2, 5}
4	{0, 3, 13, 15}
5	{0, 1, 24, 37, 41}

b) Utilisation de séquences connues

L'idée ici est de se ramener à des résultats déjà établis. Les ensembles simplement orthogonaux ont fait l'objet de nombreuses recherches. On retrouve dans la littérature différentes appellations : "séquences B-2", "séquences de Sidon" [52], "règles de Golomb" [53],... qui sont autant de variantes d'un même problème.

Les règles de Golomb retiennent particulièrement notre attention. En effet celles-ci représentent un ensemble de nombres tels que toutes les différences (sommes) simples d'éléments de cet ensemble soient distinctes. De plus divers algorithmes (SHIFT, GARSP, ...) permettent l'obtention d'ensembles aux longueurs réduites.

En fait, il apparaît qu'une telle idée est difficile à mettre en oeuvre. Pour que ce principe soit valable il faut à partir d'un ensemble de valeurs $G = \{0, g_1, g_2, ..., g_{J-1}\}$ considérer l'ensemble formé par toutes les sommes simples : $G_1 = \{g_1, g_1+g_0, ..., g_{J-2} + g_{J-1}\}$ et essayer de faire correspondre l'ensemble des valeurs de G_1 avec une séquence de règle de Golomb connue. De cette façon on serait sûr que toutes les sommes doubles de G (sommes simples de G_1) seraient distinctes. On se heurte là à deux problèmes :

. la non-certitude de pouvoir identifier une séquence de règles de Golomb à l'ensemble G (il faut s'assurer que l'on puisse bien repasser des sommes aux valeurs des g_i).

. la dépendance de l'ordre des sommes simples en fonction de l'ordre des valeurs de g_i (de cet agencement dépend l'identification à une séquence de règles de Golomb).

Ceci ajouté à la croissance rapide du nombre de sommes simples en fonction du nombre d'éléments de l'ensemble de départ rend cette technique difficilement applicable.

Considérons un petit exemple concernant des ensembles sans les négatifs pour illustrer ces propos. On connaît une suite de règles de Golomb r_i tels que : $r_i + r_j$ soient distincts avec $i > j$. Notre but étant d'obtenir un ensemble de générateurs g_i tels que $g_i - g_j + g_k - g_l$ soient distincts avec $i > j$ et $k > l$ on va essayer d'identifier r_m avec $g_p - g_q$. Supposons que l'on cherche un total de 3 générateurs $\{g_0, g_1, g_2\}$. Cet ensemble va générer 3 différences simples positives ($\frac{J(J-1)}{2}$ dans le cas général). Il nous faut donc choisir l'identification à une séquence existante comprenant 3 nombres différents de 0. La plus grande différence obtenue au sein de notre ensemble de générateurs ordonné sera naturellement $g_2 - g_0$. Dans l'optique de minimiser g_2 on va essayer la suite de règles de Golomb minimale {0, 1, 4, 6} à laquelle on retire le "0". Ainsi on aimerait avoir : $g_2 - g_0 = 6$, $g_2 - g_1 = 4$ *ou* 1 et $g_1 - g_0 = 1$ *ou* 4. Un bref calcul indique que cette situation n'est pas possible. Ainsi, il faut tester tous les ordonnancements de nos générateurs possibles mais en plus, il faut tester diverses règles de Golomb pour en trouver une qui convienne, ce qui n'est pas simple.

Il est également envisageable d'aborder cette méthode sous un angle différent en adaptant la méthode GARSP ([48]) comme ceci est présenté à l'Annexe III.

4.2.4 Comparaisons des résultats

A ce stade-ci de l'étude il serait vain de se lancer dans des comparaisons très poussées car, ne l'oublions pas, ce sont les générateurs après réduction qui nous préoccupent. Cependant plus l'ensemble à réduire sera de courte longueur plus l'on peut espérer obtenir un bon résultat. En effet même si ce sont les dernières unités pour se rapprocher du minimum réel qui sont les plus dures à gagner, plus l'on part de loin et plus l'on aura de travail de réduction à fournir.

La recherche exhaustive fournit naturellement les meilleurs résultats envisa-

geables mais, on l'a vu, son application ne se limite qu'à de faibles valeurs de J. Supposons en effet que l'on s'intéresse au cas $J = 8$ et supposons le plus grand élément égal à 364. On a alors un ensemble de la forme: $\{0, g_0, ..., g_5, g_6, 364\}$. Chaque g_i peut donc varier entre $g_{i-1} + 1$ et $364 - (7 - g_i)$. Ce qui donne le nombre de possibilités suivant:

$$\sum_{g_1=1}^{364-6} \sum_{g_2=g_1+1}^{364-5} \sum_{g_3=g_2+1}^{364-4} \sum_{g_4=g_3+1}^{364-3} \sum_{g_5=g_4+1}^{364-2} \sum_{g_6=g_5+1}^{364-1} = 3{,}048{,}385{,}428{,}726\ (3\ 10^{12}) \qquad (4.2)$$

Si l'on multiplie ce résultat par le temps de calcul nécessaire aux différents tests, il est clair que cette méthode devient inutilisable. Cependant les valeurs obtenues pour J petit donnent une référence pour comparer les autres techniques mises en jeu.

Deux tableaux (4.8 et 4.9) résument les différents résultats obtenus. Il est à noter que la comparaison avec la géométrie projective n'est pas tout à fait "équitable". En effet, la génération pseudo-aléatoire, de part le fait qu'on effectue un certain nombre d'essais pour ne conserver que le meilleur résultat, présente intrinsèquement un effort de réduction. Ce n'est pas le cas de la géométrie projective. Une comparaison plus significative après réduction sera présentée par la suite (section 4.5). Il convient cependant de garder à l'esprit que le temps requis par la génération par géométrie projective est très largement supérieur à celui nécessaire dans le cas d'une génération pseudo-aléatoire.

- ensembles sans les négatifs

Tableau 4.8 – *Comparaisons des méthodes de génération pour des ensembles sans les négatifs*

J	longueur de l'ensemble		
	rech. exh.	pseudo-aléa.	géo. proj.
2	1	1	1
3	4	4	18
4	12	12	69
5	33	34	242
6	-	96	625
7	-	198	1237
8	-	379	2510
9	-	1051	3723
10	-	1960	6965

Il n'est pas très significatif dans ce cas là de comparer la méthode par géométrie projective aux autres techniques. En effet on a vu que celle-ci permet de générer des ensembles complets qui possèdent plus d'équations à vérifier que les ensembles sans les négatifs.

- ensembles complets

Tableau 4.9 – *Comparaisons des méthodes de génération pour des ensembles complets*

J	longueur de l'ensemble		
	rech. exh.	pseudo-aléa.	géo. proj.
2	1	1	1
3	5	5	18
4	15	15	69
5	41	43	242
6		107	625
7	-	257	1237
8	-	579	2510
9	-	1202	3723
10	-	2193	6965

On remarque que sur les premières valeurs la stratégie de génération aléatoire n'est pas loin de fournir les valeurs minimales voire d'y arriver pour $J = 2$, 3 et 4. Ceci est en réalité normal. L'algorithme utilisé est en effet particulièrement rapide, notamment lorsque le nombre d'éléments est petit. Ceci permet sur une courte période de temps de lancer de nombreuses simulations. L'obtention du "bon" ensemble étant probabiliste (il existe nécessairement une suite de tirage particulière conduisant à la génération de la séquence minimale), plus le nombre d'essais sera important et plus la chance d'aboutir à l'ensemble minimal sera grande. Ceci explique pourquoi cette technique devient moins précise avec l'augmentation du nombre d'éléments de l'ensemble (la probabilité de tomber sur la bonne séquence est de plus en plus faible).

4.3 Réduction

4.3.1 Problématique

La section précédente nous a procuré différentes méthodes nous permettant de générer des ensembles doublement orthogonaux du type $\{0, g_1, ..., g_{J-1}\}$. On a vu également que de l'élément g_{J-1} dépendent la latence du décodage et la mémoire nécessaire au niveau du décodeur. Il est donc crucial de minimiser cette valeur au maximum pour éviter tout délai de traitement, délai, qui on l'a vu est préjudiciable dans toutes les applications. La valeur minimale de g_{J-1} sera par la suite nommée la longueur minimale (g^*_{J-1}) que l'on va tenter d'atteindre pour un nombre de générateurs J donné.

4.3.2 Recherche de g^*_{J-1}, obtention de bornes et d'estimées

4.3.2.1 Ensembles sans les négatifs

On a établi au (3.4.2.2) que dans le cas des ensembles sans les négatifs, pour un ensemble de J éléments, il y a $N_s(J) = \frac{J(J-1)}{2}$ différences simples et $N_d(J) = \frac{J(J-1)(J^2-J+4)}{12}$ différences doubles. Ci-après figurent différentes tentatives effectuées pour élaborer une théorie satisfaisante visant l'obtention de l'expression de la longueur minimale.

a) Cas idéal

On suppose que l'ensemble $\{0, g_1, ..., g_{J-1}\}$ $(0 < g_1 < ... < g_{J-1})$ est "parfait". C'est à dire que la réunion des ensembles formés par les valeurs des différences simples et doubles ne contient que des entiers consécutifs distincts. Un tel ensemble vérifierait nécessairement la condition de double orthogonalité puisque les différences doubles et simples seraient toutes distinctes et distinctes entre elles. La plus grande de ces valeurs étant $g_{J-1} - 0 + g_{J-1} - 0 = 2.g_{J-1}$, la longueur minimale de ce cas idéal serait :

$$2.g_{J-1} = \underbrace{N_d(J) + N_s(J)}_{\textit{Nombre de valeurs contenues dans l'ensemble "réunion"}} \tag{4.3}$$

$$g_{J-1} = \frac{N_d + N_s}{2} = \frac{J(J-1)(J^2-J+4)}{24} + \frac{J(J-1)}{4} \tag{4.4}$$

$$g_{J-1} = \frac{J(J-1)(J^2-J+10)}{24} \tag{4.5}$$

En pratique cette situation est impossible pour $J \geq 4$. Les différentes différences simples et doubles ne peuvent se repartir de manière à créer une succession de nombres entiers consécutifs. Ceci s'explique par le fait que modifier la valeur des différents générateurs ne permet pas d'agir indépendamment sur les différences simples d'un côté et sur les différences doubles de l'autre. Ces grandeurs sont en effet "reliées" ce qui impose certains écarts. Ainsi, l'expression obtenue, à défaut de fournir la longueur minimale, donne une borne inférieure ($B_{inf}(J) = \frac{J(J-1)(J^2-J+10)}{24}$, borne "cas idéal") de la grandeur g^*_{J-1}.

La détermination de la longueur minimale se heurte à de nombreuses difficultés. En particulier, le dénombrement pose certains problèmes. Illustrons cela par un exemple en considérant un ensemble de 4 générateurs: $\{0, g_1, g_2, g_3\}$ $(0 < g_1 < g_2 < g_3)$. Exprimer le fait que toutes les différences doubles sont distinctes revient à considérer factoriel(15) inéquations (il y a 16 différences doubles). Après étude de ces inéquations (élimination des redondances) on s'aperçoit qu'il ne reste plus que 44 inégalités. L'ordre des éléments permet de supprimer 13 autres inéquations supplémentaires pour arriver à un nombre final de 31. L'idée est alors à l'image de ce qui a été fait pour calculer les grandeurs $N_s(J)$ et $N_d(J)$ de dénombrer les possibilités de valeurs pour notre ensemble en fixant d'abord g_1 puis g_2 et g_3. En réalité, on s'aperçoit très rapidement que même pour un ensemble avec peu d'éléments cette opération s'avère extrêmement fastidieuse et inutilisable.

b) Relation de récurrence

Puisque l'on a été capable de déterminer la borne par une recherche exhaustive pour un faible nombre d'éléments, il serait très intéressant de déterminer s'il existe une relation de récurrence permettant de passer de la longueur minimale pour un

ensemble de J générateurs à celle valable pour un ensemble de $J+1$ générateurs. Cette recherche n'a pas permis d'aboutir à la longueur minimale comme on l'aurait souhaité mais une borne supérieure fut cependant obtenue (notation $B_{sup}(J)$) :

$$[B_{sup}(J+1) = 3.B_{sup}(J) + 1] \; avec \; B_{sup}(4) = 12 \tag{4.6}$$

L'appellation "borne supérieure" se justifie par le comportement asymptotique de celle-ci. En effet $B_{sup}(J)$ évolue comme 3^J qui croît de manière exponentielle avec J.

Prouvons cette équation de récurrence :

- $B_{sup}(4) = 12$

En se reportant aux résultats présentés page 71, on note que pour $J = 14$ l'ensemble obtenu $\{0, 3, 11, 12\}$ a une longueur maximale égale à 12. Ceci assure que la valeur 12 est bien une borne supérieure pour un ensemble complet comportant 4 éléments.

- $B_{sup}(4) \rightarrow B_{sup}(5)$

Supposons $B_{sup}(5) = 3 * B_{sup}(4) + 1$ soit $B_{sup}(5) = 3 * 12 + 1 = 37$. En se reportant une nouvelle fois aux résultats de la page 71, on remarque que l'ensemble contenant 5 éléments présenté $\{0, 2, 23, 28, 34\}$ a une longueur maximale inférieure à 37. Ceci fait donc de $B_{sup}(5)$ ainsi définie une borne supérieure.

- $B_{sup}(J) \rightarrow B_{sup}(J+1)$

Considérons notre ensemble $\{0, g_1, ..., g_{J-1}\}$ avec $g_{J-1} = B_{sup}(J)$. On sait que les différences simples sont comprises entre 0 et g_{J-1} et les différences doubles entre 0 et $2.g_{J-1}$. Supposons que l'on se contente de construire g_J à la suite de cet en-

semble avec $g_J = 3.g_{J-1} + 1$. Les nouvelles différences simples engendrées par g_J seront comprises entre $g_J - g_{J-1}$ et g_J et les nouvelles différences doubles entre $g_J - g_{J-1} + g_J - g_{J-1}$ et $g_J + g_J$. Comme $g_J = 3.g_{J-1} + 1$ on aura:

$g_J - g_{J-1} > 2.g_{J-1}$, $g_J > 3.g_{J-1}$, $g_J - g_{J-1} + g_J - g_{J-1} > 4.g_{J-1}$, $g_J + g_J > 6.g_{J-1}$.

On est ainsi certain que les nouvelles différences obtenues par l'ajout de g_J ne seront pas égales à des valeurs de différences générées par l'ensemble $\{0, g_1, ..., g_{J-1}\}$. On obtiendra bien de la sorte un ensemble de $J + 1$ éléments doublement orthogonal. La grandeur $B_{sup}(J + 1) = g_J = 3.B_{sup}(J) + 1$ est donc une borne supérieure ce qui achève de prouver la relation de récurrence.

c) Autres résultats

Diverses tentatives ont permis d'obtenir d'autres bornes malheureusement difficilement justifiables et vérifiables. Citons en particulier l'expression $\frac{1}{5}A_J^4$. Théoriquement il semble que la détermination de la borne soit un problème extrêmement complexe qui n'est d'ailleurs pas résolu dans le cas des ensembles simplement orthogonaux. Malgré tout, ces mises en forme (illustrées par le tableau 4.10) permettent d'obtenir des ordres de grandeurs et de mieux apprécier le comportement de cette valeur limite.

Tableau 4.10 - *Comportement des bornes "cas idéal" et "récursive" en fonction de J pour des ensembles sans les négatifs*

J	borne "cas idéal" ($B_{inf}(J)$)	"récursive" ($B_{sup}(J)$)
2	1	-
3	4	-
4	11	12
5	25	37
6	50	112
7	91	337
8	154	1012
9	246	3037
10	375	9112

L'obtention de la longeur minimale de manière théorique n'a donc pas pu être établie. Ceci est regrettable car cela eût été un apport considérable pour générer des ensembles minimaux. Cela n'a cependant rien d'étonnant vu la complexité du problème.

4.3.2.2 Ensembles complets

On a vu que les ensembles complets en imposant plus de conditions aux générateurs, donnent lieu à de meilleures performances. L'étude de la longueur minimale pour ces ensembles se heurte naturellement aux mêmes problèmes que dans le cas précédent.

a) Cas idéal

Identiquement à ce qui a été effectué à la section précédente, on suppose que notre ensemble de générateurs est tel que la réunion des valeurs des différences simples et doubles forme un ensemble d'entiers consécutifs. La valeur maximale de cet ensemble est naturellement $g_{J-1} - 0 + g_{J-1} - 0 = 2.g_{J-1}$, celle minimale est $0 - g_{J-1} + 0 - g_{J-1} = -2.g_{J-1}$. En sachant (3.4.2.2) que le nombre de différences simples et doubles sont respectivement: $N_s(J) = J(J-1)$ et $N_d(J) = \frac{1}{4}J(J-1)(J^2 - J + 2)$ on obtient:

$$2.g_{J-1} - (-2.g_{J-1}) = \underbrace{N_d(J) + N_s(J)}_{\textit{Nombre de valeurs contenues dans l'ensemble "réunion"}} \tag{4.7}$$

$$4.g_{J-1} = J(J-1) + \frac{1}{4}J(J-1)(J^2 - J + 2) \tag{4.8}$$

$$g_{J-1} = \frac{1}{16}J(J-1)(J^2 - J + 6) \tag{4.9}$$

Une fois encore cette grandeur est une borne inférieure par rapport à ce qui est obtenu en pratique (le cas idéal n'est jamais atteint pour $J \geq 5$ pour les mêmes raisons que dans le cas des ensembles sans les négatifs).

b) Imbrication

Une autre piste envisagée fut celle consistant à examiner les sous ensembles complets dont est constitué un ensemble de départ. Prenons par exemple un ensemble de 6 éléments du type $\{0, g_1, g_2, g_3, g_4, g_5\}$. Connaissant les bornes des ensembles dont le nombre d'éléments est 2, 3, 4, 5 par recherche exhaustive, peut-on en déduire quelque chose sur la borne de l'ensemble contenant 6 générateurs?

. En se basant sur la longueur minimale (5) d'un sous-ensemble de 3 éléments on a : $g_2 \geq 5$, $g_3 - g_1 \geq 5$, $g_4 - g_2 \geq 5$ et $g_5 - g_3 \geq 5$.

. De même pour les sous-ensembles de taille 4 (longueur minimale 15) : $g_3 \geq 15$, $g_4 - g_1 \geq 15$ et $g_5 - g_2 \geq 15$.

. Pour les sous-ensembles comportant 5 éléments (longueur minimale 41) : $g_4 \geq 41$ et $g_5 - g_1 \geq 41$.

En regroupant ces conditions, on obtient :

$$\begin{cases} g_2 \geq max(5, g_1 + 1) \\ g_3 \geq max(15, g_1 + 5, g_2 + 1) \\ g_4 \geq max(41, g_1 + 15, g_2 + 5, g_3 + 1) \\ g_5 \geq max(g_1 + 41, g_2 + 15, g_3 + 5, g_4 + 1) \end{cases} \tag{4.10}$$

On a vu au chapitre 3 (3.4.2.1) qu'il existait toujours au moins deux ensembles convenant dont l'un est obtenu à partir de l'autre par une simple symétrie présentée

par l'équation 3.9. On peut donc imposer une condition supplémentaire pour "centrer" nos générateurs. Par exemple pour l'ensemble $\{0, g_1, ..., g_5\}$, on peut imposer $g_2 \leq \lfloor \frac{g_5}{2} \rfloor$. On obtient alors le nouveau lot de conditions :

$$\begin{cases} min(g_3 - 1, \lfloor \frac{g_5}{2} \rfloor) \geq g_2 \geq max(5, g_1 + 1) \\ g_3 \geq max(15, g_1 + 5, g_2 + 1) \\ g_4 \geq max(41, g_1 + 15, g_2 + 5, g_3 + 1) \\ g_5 \geq max(g_1 + 41, g_2 + 15, g_3 + 5, g_4 + 1) \end{cases} \tag{4.11}$$

Malheureusement, ces conditions n'apportent pas suffisamment de précisions pour pouvoir tirer quelques conclusions que ce soient sur g_5.

c) Espaces libres

Il a été entrepris de rechercher une relation entre la longueur minimale d'un ensemble de taille J et le nombre d'entiers non atteints par les valeurs des différences doubles. Il est légitime de penser qu'il puisse y avoir un rapport, ce qui traduirait l'écart de l'ensemble obtenu à un ensemble "parfait". On a vu en effet que dans ce dernier cas idéal, la réunion des différences simples et doubles regroupe des entiers consécutifs. Ici l'on ne considère que les différences doubles par commodité mais l'on sait que le nombre de différences simples est de $J(J-1)$. Un simple décalage suffit alors pour l'étude de la réunion.

Ci-dessous figure le dénombrement utilisé pour les cas dont la longueur minimale est établie de manière sûre (recherche exhaustive), les entiers non utilisés sont soulignés :

- $J = 2$, ensemble $\{0, 1\}$

  $\{0, \underline{1}, 2\}$

On compte 1 entier manquant.

- $J = 3$, ensemble $\{0, 2, 5\}$

  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

  On dénombre 4 entiers manquants.

- $J = 4$, ensemble $\{0, 1, 11, 15\}$.

  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 , 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30}

  On repère ainsi 9 entiers manquants.

- $J = 5$, ensemble $\{0, 1, 24, 37, 41\}$

  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82}

  27 entiers manquants apparaissent ici.

- $J = 6$, ensemble $\{0, 1, 17, 70, 95, 100\}$

  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37 , 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54 , 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84 , 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169,

170, 171, 172, 173 , 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200}

On répertorie ici 80 entiers manquants.

On peut alors espérer trouver une relation polynômiale reliant J au nombre d'entiers manquants. A la vue des résultats précédemment obtenus, il semble légitime de rechercher un polynôme de degré 4 (le nombre d'équations est de degré 4, on considère des différences impliquant 4 nombres...). On note celui-ci $n(J) = a.J^4 + b.J^3 + c.J^2 + d.J + e$

On arrive alors au système suivant :

$$\begin{pmatrix} 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 3^4 & 3^3 & 3^2 & 3^1 & 3^0 \\ 4^4 & 4^3 & 4^2 & 4^1 & 4^0 \\ 5^4 & 5^3 & 5^2 & 5^1 & 5^0 \\ 6^4 & 6^3 & 6^2 & 6^1 & 6^0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} = \begin{pmatrix} 1 \\ 4 \\ 9 \\ 27 \\ 80 \end{pmatrix} \tag{4.12}$$

Soit :

$$\begin{pmatrix} 16 & 8 & 4 & 2 & 1 \\ 81 & 27 & 9 & 3 & 1 \\ 256 & 64 & 16 & 4 & 1 \\ 625 & 125 & 25 & 5 & 1 \\ 1296 & 216 & 36 & 6 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} = \begin{pmatrix} 1 \\ 4 \\ 9 \\ 27 \\ 80 \end{pmatrix} \tag{4.13}$$

La résolution de ce système aboutit alors à :

$$a = \frac{11}{24} \ , \ b = \frac{-55}{12} = -11024 \ , \ c = \frac{409}{24} \ , \ d = \frac{-299}{12} = -59824 \ , \ e = 12 = \frac{288}{24} \tag{4.14}$$

On peut alors supposer une mise en forme du nombre d'éléments manquants $n(J)$ sous la forme :

$$n(J) = \frac{11}{24}J^4 - \frac{110}{24}J^3 + \frac{409}{24}J^2 - \frac{598}{24}J + \frac{288}{24} \tag{4.15}$$

$$n(J) = \frac{J-1}{24}(11J^3 - 99J^2 + 310J - 288) \tag{4.16}$$

A partir de là il est possible d'obtenir une approximation. En effet en additionnant le nombre d'entiers manquants et le nombre de différences doubles positives on est censé obtenir deux fois la valeur maximale de l'ensemble. On établit ainsi l'approximation $(B_{appr}(J))$ du type :

$$B_{appr}(J) = \frac{1}{2}(\frac{J-1}{24}(11J^3 - 99J^2 + 310J - 288)) + \frac{1}{8}(J(J-1)(J^2 - J + 2)) \tag{4.17}$$

$$B_{appr}(J) = \frac{1}{24}(J-1)(7J^3 - 51J^2 + 158J - 144) \tag{4.18}$$

Cette grandeur est naturellement à considérer avec beaucoup de précautions du fait qu'elle est générée à partir d'une extrapolation polynômiale. Cependant à la vue des résultats obtenus après quelques calculs, celle ci ne s'avère pas fortement éloignée. Pour $J \leq 10$, elle représente une approximation par valeurs inférieures et à partir de 10 par valeurs supérieures.

d) équivalence différences - sommes

On a établi (3.4.2.1) l'équivalence entre l'obtention de différences doubles distinctes et de sommes doubles distinctes. La condition de double orthogonalité s'exprime alors :

$$\forall(p, q, r, s, p', q', r', s') \textit{ tel que}$$
$$(p, q, r, s) \textit{ ne soit pas une permutation de } (p', q', r', s')$$

$$g_p + g_q + g_r + g_s \neq g_{p'} + g_{q'} + g_{r'} + g_{s'}. \tag{4.19}$$

On peut se servir de cette équivalence pour élaborer une nouvelle borne inférieure ($B_{inf1}(J)$) suivant un mode équivalent à celui établi pour la "borne cas idéal". On sait qu'il existe A_J^4 sommes de quatre éléments parmi un ensemble de J éléments. On peut, sans tenir compte des restrictions imposées sur l'équation, identifier ce nombre au nombre de sommes doubles possibles dans notre ensemble doublement orthogonal considéré (il y en a en réalité un peu plus du fait de la possibilité pour un élément de se répéter plus d'une fois). En supposant que les valeurs des sommes obtenues forment un ensemble idéal, on peut fixer $B_{inf1}(J) = \frac{1}{4}A_4^J$. Pour $J > 4$, les quelques sommes que notre ensemble orthogonal peut générer de plus sont largement compensées par le fait qu'il est impossible d'obtenir un ensemble "parfait".

e) Autres tentatives

D'autres essais d'obtention de la longueur minimale ont été réalisés. On citera par exemple, une tentative prenant en compte le nombre d'apparitions d'un générateur dans l'expression des sommes ou des dénombrements plus complexes.

Cependant les techniques présentées auparavant ont abouti à de meilleurs résultats.

Une comparaison des grandeurs obtenues est reportée dans le tableau 4.11.

Tableau 4.11 - *Comportement des bornes "cas idéal ($B_{inf}(J)$)", "espaces libres ($B_{appr}(J)$)" et "sommes ($B_{inf1}(J)$)" en fonction de J.*

J	borne cas idéal ($B_{inf1}(J)$)	entiers manquants ($B_{appr}(J)$)	sommes ($B_{inf1}(J)$)
4	13.5	15	6
5	32.5	41	30
6	67.5	100	90
7	126	216	210
8	217	420	420
9	351	750	756
10	540	1251	1260
11	797.5	1975	1980
12	1138.5	2981	2970
13	1579.5	4335	4290
14	2138.5	6110	6006
15	2835	8386	8190

Il apparait très nettement que pour $J \leq 6$ B_{inf1} est des deux bornes inférieures la plus intéressante puisque la plus élevée. B_{appr} est très proche de B_{inf1} au point de l'égaler pour $J = 8$. Leurs positions respectives s'échangent d'ailleurs pour cette valeur. Une étude polynômiale de la différence des deux indique un changement de signe pour les valeurs de $J = 8$ et $J \approx 12$.

4.3.3 Théorie - techniques utilisées

Les différentes expérimentations proposées ont prouvé que la détermination de la longueur minimale de manière théorique est problématique. Même si les expressions obtenues sont intéressantes car elles donnent un ordre de grandeur de la longueur minimale, il n'en demeure pas moins qu'il sera plus difficile d'obtenir l'ensemble le

plus réduit possible sans la connaissance de cette grandeur.

Partant d'un ensemble précédemment généré grâce à une des méthodes vues précédemment (4.2), il nous faut trouver une technique permettant de réduire cet ensemble tout en lui conservant sa propriété de double orthogonalité. Vu les difficultés rencontrées pour l'établissement de la longueur minimale, on se doute qu'il en sera de même pour la génération d'un ensemble minimal.

On remarquera tout d'abord que les opérations élémentaires ne modifient pas le caractère doublement orthogonal d'un ensemble. Partons d'un ensemble doublement orthogonal $\{0, g_1, g_2, ..., g_{J-1}\}$ qui vérifie la condition 3.4.2. Supposons que l'on multiplie par un entier r non nul, il est clair alors que si l'on considère l'ensemble $\{0, r.g_1, r.g_2, ..., r.g_{J-1}\}$, il vérifie les conditions 3.4.2. De même l'ensemble formé par la somme de chaque générateur avec un nombre p $\{p, g_1+p, g_2+p, ..., g_{J-1}+p\}$ est encore doublement orthogonal.

a) Modulo

Cette méthode a été proposée initialement par François Gagnon et David Haccoun [13] pour faire suite à la technique de génération par géométrie projective. Celle ci on l'a vu, génère des ensembles doublement orthogonaux modulo un entier n ((α) d'ordre n). Naturellement plus le modulo est petit et plus la longueur de l'ensemble diminue. L'idée est alors de tenter de réduire cette grandeur tout en modifiant l'ensemble.

De façon générale, considérons notre ensemble $\{0, g_1, ..., g_{J-1}\}$ doublement

orthogonal. On a donc :

$$g_i - g_j + g_k - g_l \neq g_{i'} - g_{j'} + g_{k'} - g_{l'} \tag{4.20}$$

Si n entier est tel que l'inéquation précédente soit encore vérifiée modulo(n) on arrive à :

$$g_i - g_j + g_k - g_l \ mod(n) \neq g_{i'} - g_{j'} + g_{k'} - g_{l'} \ mod(n) \tag{4.21}$$

en multipliant par r de chaque côté de cette inéquation tel que r et n soient premiers entre eux ($r \bigwedge n = 1$) on arrive à

$$r.g_i - r.g_j + r.g_k - r.g_l \ mod(n) \neq r.g_{i'} - r.g_{j'} + r.g_{k'} - r.g_{l'} \ mod(n) \tag{4.22}$$

Ce qui équivaut à :

$$\begin{gathered}(r.g_i + d) - (r.g_j + d) + (r.g_k + d) - (r.g_l + d) \ mod(n) \\ \neq \\ (r.g_{i'} + d) - (r.g_{j'} + d) + (r.g_{k'} + d) - (r.g_{l'} + d) \ mod(n)\end{gathered} \tag{4.23}$$

En appliquant la propriété : $(a + b) \ mod(n) = (amod(n) + bmod(n)) \ mod(n)$ on arrive à :

$$\begin{gathered}((r.g_i + d)mod(n) - (r.g_j + d)mod(n) + (r.g_k + d)mod(n) - (r.g_l + d)mod(n)) \\ mod(n) \\ \neq \\ ((r.g_{i'} + d)mod(n) - (r.g_{j'} + d)mod(n) + (r.g_{k'} + d)mod(n) - (r.g_{l'} + d)mod(n)) \\ mod(n)\end{gathered} \tag{4.24}$$

En définissant $a_i = (r.g_j + d) mod(n)$ on a:

$$a_l - a_p + a_s - a_e \ mod(n) \neq a_{l'} - a_{p'} + a_{s'} - a_{e'} \ mod(n) \tag{4.25}$$

Ce qui assure bien entendu que:

$$a_l - a_p + a_s - a_e \neq a_{l'} - a_{p'} + a_{s'} - a_{e'} \tag{4.26}$$

Ainsi avec les restrictions suivantes:

- $g_i - g_j + g_k - g_l$ distincts modulo n
- r (non nul) et n premiers entre eux

L'ensemble $\{a_i\}$ défini à partir de $\{g_j\}$ doublement orthogonal tel que $a_l = (r.g_p + d) mod(n)$ est doublement orthogonal.

La mise en œuvre de cette méthode est la suivante:

1. On considère l'ensemble en cours et on choisit un entier qui représentera le modulo.
2. On vérifie que les différences doubles demeurent distinctes après application du modulo.
3. On effectue des opérations élémentaires sur l'ensemble considéré (addition et multiplication par un nombre compris entre 0 (1 pour la multiplication) et la valeur du modulo-1. Ceci pour toutes les valeurs possibles en s'assurant que le facteur multiplicatif et le modulo soient premiers entre eux. Si l'on obtient un ensemble dont la longueur est inférieure à celle de départ, on conserve cet ensemble et on recommence la procédure à l'étape 2. Sinon on décrémente le modulo d'une unité et on recommence à l'étape 2.

Une représentation schématique de cette procédure est présentée à la figure 4.2:

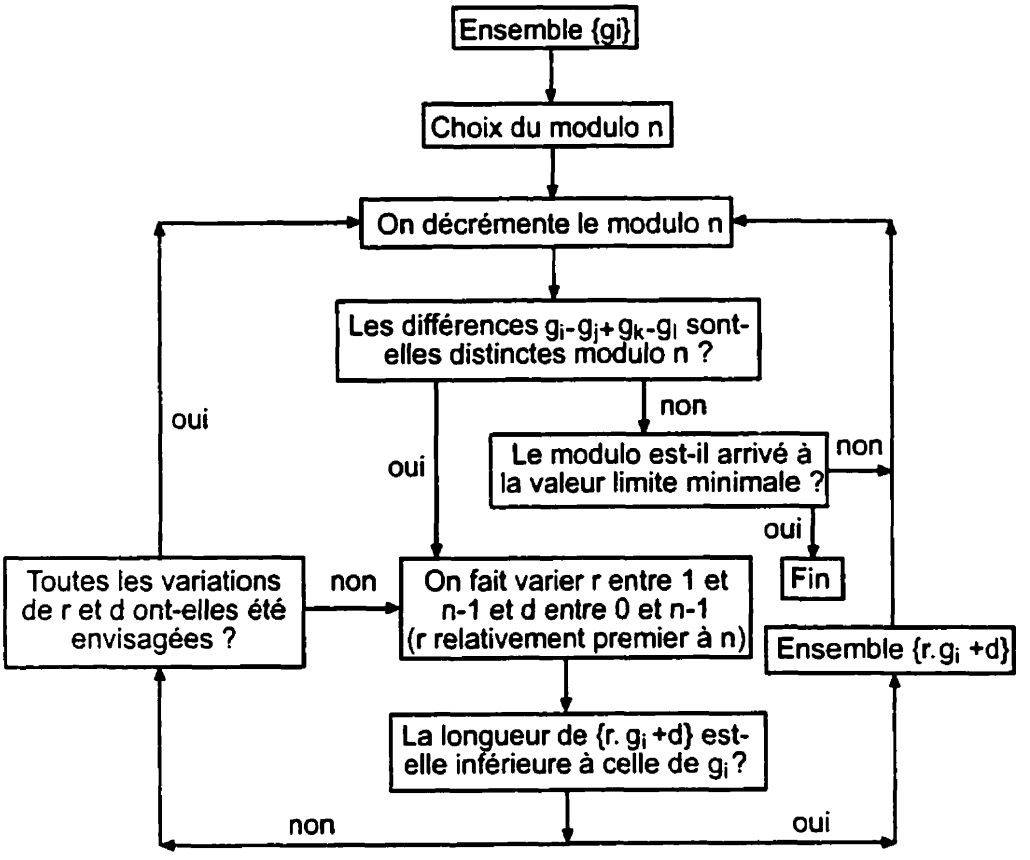

Figure 4.2 - *Processus de réduction par méthode de modulo*

Il est possible d'effectuer diverses variations sur ce thème qui furent tour à tour essayées :

. Le nouvel ensemble obtenu ne sera validé que si la borne obtenue est inférieure à la précédente moins un certain paramètre qu'il est possible de faire évoluer.

. On considère toujours le modulo initial.

. Lorsqu'on ne trouve plus d'ensemble, on rehausse la valeur du modulo.

b) Transformation

L'idée est de passer d'un ensemble doublement orthogonal $\{0, g_1, ..., g_{J-1}\}$ à

un autre ensemble $\{0, a_1, ..., a_{J-1}\}$ possédant la même propriété mais de longueur moindre par une transformation nommée "f". Ceci se formalise sous la forme (de manière triviale f se doit d'être injective) :

$\forall(p, q, r, s, p', q', r', s')$ *tel que* $p \neq q$, $p \neq s$, $q \neq r$, $r \neq s$, $p \geq r$, $q \geq s$,
$p' \neq q'$, $p' \neq s'$, $q' \neq r'$, $r' \neq s'$, $p' \geq r'$, $q' \geq s'$ *et* $(p, q, r, s) \neq (p', q', r', s')$.

$$g_p - g_q + g_r - g_s \neq g_{p'} - g_{q'} + g_{r'} - g_{s'}$$

$$\equiv$$

$$f(a_i) - f(a_j) + f(a_k) - f(a_l) \neq f(a_{i'}) - f(a_{j'}) + f(a_{k'}) - f(a_{l'}).$$

En supposant l'hypothèse simplificatrice de f linéaire, on arrive à

$$f((a_i - a_j + a_k - a_l) - (a_{i'} - a_{j'} + a_{k'} - a_{l'})) \neq 0.$$

Une condition suffisante pour être sûr d'obtenir $a_i - a_j + a_k - a_l \neq a_{i'} - a_{j'} + a_{k'} - a_{l'}$ pour $(i, j, k, l) \neq (i', j', k', l')$ est de choisir f telle que $f(i) \neq 0$ pour $i \neq 0$. On arrive alors à rechercher les applications du type :

$$\begin{cases} f(i) \neq 0 \text{ pour } i \neq 0 \\ f \text{ injective} \\ f \text{ linéaire} \\ \text{espace d'arrivée plus petit que celui de départ} \end{cases} \tag{4.27}$$

Malheureusement il existe de nombreuses applications de ce type et rien de bien concret n'a pu ressortir de cette étude.

c) Stationnarité

Cette technique se base sur l'étude d'un espace de J éléments pour générer un espace de $J - 1$ éléments. La procédure qui a donné des résultats intéressants est la suivante :

1. On part d'un ensemble de J éléments doublement orthogonal généré par une

des méthodes illustrées.

2. On retire l'élément qui par ordre croissant se trouve juste après le zéro. On se retrouve alors avec un ensemble de $J-1$ éléments, soit g_{J-2} son plus grand élément.
3. On effectue une recherche exhaustive pour trouver le plus petit élément supérieur à g_{J-2} qui permettrait de compléter notre ensemble doublement orthogonal à J élément.
4. Ceci fait, on stocke la différence entre le plus grand élément de cet ensemble et le plus petit élément non nul.
5. On recommence la procédure à l'étape 2
6. On remarque qu'au bout d'un certain nombre d'itérations, la valeur de la différence calculée reste constante. On s'aperçoit expérimentalement que cette valeur dépend de l'ensemble de départ. Il nous suffit alors de considérer le dernier ensemble obtenu, d'y retirer le zéro et de soustraire à chaque élément la valeur du plus petit élément de cet ensemble pour obtenir un ensemble de $J-1$ éléments réduit.

Illustrons ce cheminement par un petit exemple:

Tableau 4.12 – *Exemple de procédure de stationnarité pour 3 éléments*

étapes	ensembles	$g_{J-1}-g_1$
initialement	{0, 1, 5, 21}	20
déroulement	{0, 5, 21, 26}	17
	{0, 21, 26, 27}	5
	{0, 26, 27, 31}	5
	{0, 27, 31, 32}	5
	{0, 31, 32, 36}	5
fin	{0, 1, 5}	

Cette procédure rapide donne de bons résultats qui malheureusement se dégradent

rapidement avec une augmentation du nombre d'éléments.

d) Autres méthodes

La réduction ne pouvant s'opérer de manière exacte quelques tâtonnements ont permis d'améliorer les résultats. Il serait vain de les présenter tous ici mais ils ont parfois permis d'améliorer sur une ou deux valeurs les résultats établis.

4.4 Meilleurs résultats obtenus

Les tableaux 4.13 et 4.14 répertorient les meilleurs résultats générés. D'une manière générale ils furent principalement obtenus en combinant une méthode de génération pseudo-aléatoire et une réduction basée sur l'utilisation d'un modulo. Il est clair que plus la longueur de l'ensemble est grande et plus le temps de calcul nécessaire à l'obtention d'une bonne réduction augmente. S'il ne faut pas plus que quelques minutes pour arriver à la valeur minimale absolue pour $J = 6$, les valeurs présentées pour $J = 10$ ont été obtenues avec des expérimentations d'une durée supérieure à une semaine. Les calculs ont été "poussés plus loin" (jusqu'à $J = 15$) dans le cas des ensembles complets qui rappelons le possèdent un niveau de performance plus intéressant.

Tableau 4.13 - *Meilleurs générateurs obtenus pour des ensembles sans les négatifs*

J	générateurs
2	{0, 1}
3	{0, 1, 4}
4	{0, 3, 11, 12}
5	{0, 1, 4, 24, 33}
6	{0, 2, 53, 58, 73, 79}
7	{0, 3, 18, 119, 132, 170, 174}
8	{0, 1, 4, 40, 258, 283, 366, 375}
9	{0, 28, 33, 72, 568, 653, 829, 859, 913}
10	{0, 29, 40, 43, 1020, 1328, 1495, 1606, 1696, 1698}

Tableau 4.14 - *Meilleurs générateurs obtenus pour des ensembles complets*

J	générateurs
2	{0, 1}
3	{0, 2, 5}
4	{0, 4. 14. 15}
5	{0,1,24, 37, 41}
6	{0, 1, 17, 70, 95, 100}
7	{0, 1, 53, 128, 207, 216, 222}
8	{0, 43, 139, 322, 422, 430, 441, 459}
9	{0, 9, 21, 395, 584, 767, 871, 899, 912}
10	{0, 29, 40, 43, 1020, 1328, 1495, 1606, 1696, 1698}
11	{0, 220, 521, 695, 908, 926, 1059, 2457, 3367, 3458, 3490}
12	{0,48,212,1014,1381,2217,4198,4373,4766,4885,4914,5173}
13	{0,335,594,639,1476,1778,3034,5637,6584,9682,9934,10138,10201}
14	{0,575,1253,4863,6407,6660,7285,8443,11884,13891,15065,15484,15550, 16285}
15	{0,576,1331,1897,2222,4684,6502,7293,10785,18999,20901,22157,24372, 27107,29532}

4.5 Simulation des codes générés

Les simulations des codes ont été réalisés par Christian Cardinal qui a conçu un simulateur de décodage itératif utilisant les codes doublement orthogonaux. En agissant sur les facteurs pondérant les équations de parité vu au (3.3.2.3) celui ci est en mesure de compenser dans une certaine mesure la corrélation qui apparaît lorsque le nombre d'itérations augmente. Ceci revient en fait à parfaire les codes au sens large qui, on l'a vu (3.4.2.3), présentent des permutations identiques inévitables. La simulation a porté sur un nombre de 6 itérations (ce qui est en pratique largement suffisant).

Ces simulations furent l'occasion de comparer les codes doublement orthogonaux générés par les inventeurs de ces codes et ceux générés lors de notre étude et qui bénéficient d'une longueur grandement réduite. La comparaison entre les deux indique des performances extrêmement similaires avec un infime avantage pour les derniers codes obtenus. Le niveau de performance est proche de celui des codes turbo sans pour autant l'égaler. Les courbes 4.3 et 4.4 illustrent ces propos.

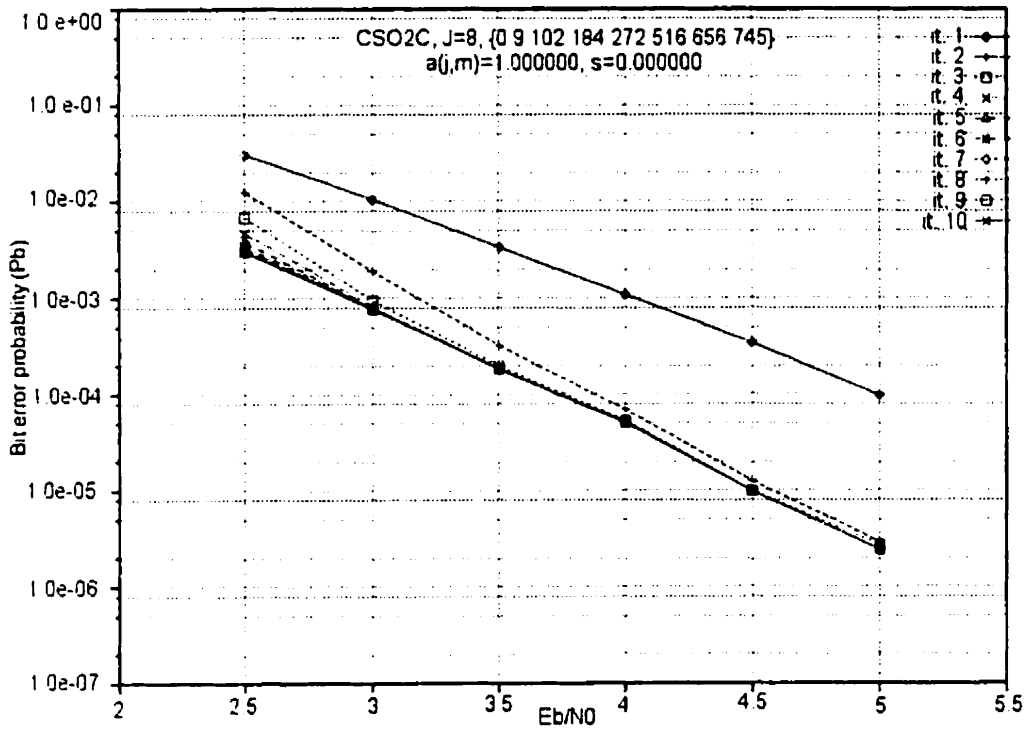

Figure 4.3 – *Probabilité d'erreur par bit en fonction de* $\frac{Eb}{No}$ *pour* $J = 8$ *(longueur non réduite)*

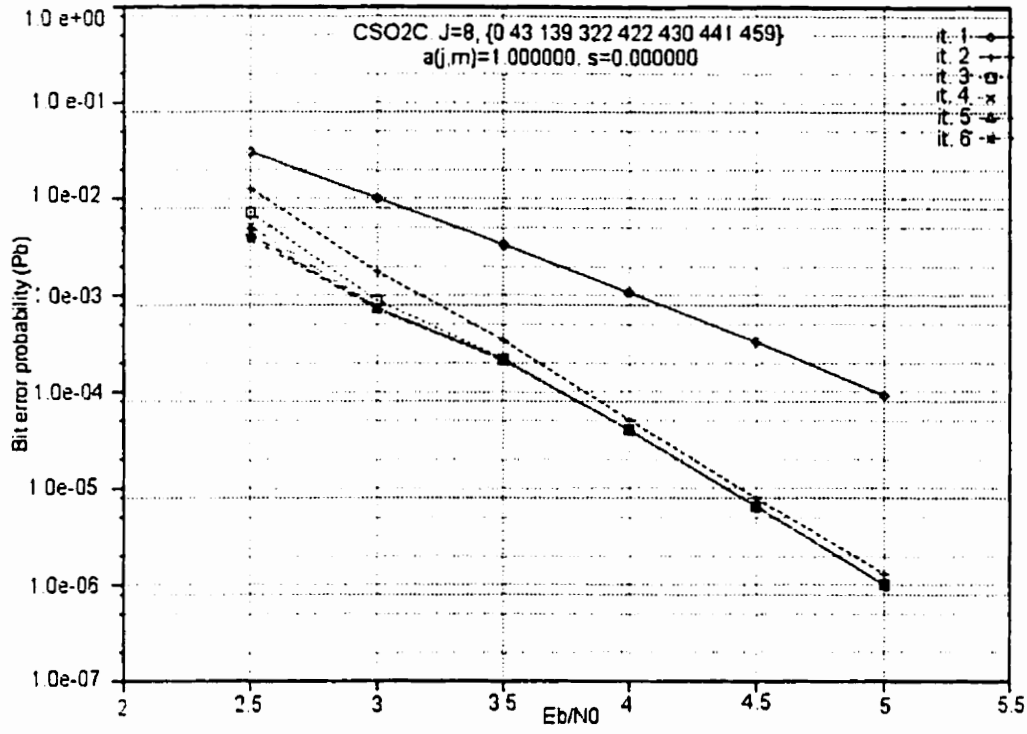

Figure 4.4 - *Probabilité d'erreur par bit en fonction de* $\frac{Eb}{No}$ *pour* $J = 8$ *(longueur réduite)*

Le fait que les deux courbes soient très similaires est significatif. En effet, cela démontre que la probabilité d'erreur en fonction du rapport signal à bruit n'est en aucune façon influencée par la longueur du code utilisé mais par la valeur de J. Ceci est intuitif puisqu'à priori cette grandeur va dépendre des propriétés du code qui ici sont les mêmes dans les deux cas.

Pour illustrer l'influence du réglage des gains au sein des équations de parité on présente les deux courbes 4.5 et 4.6 (notations issues du travail de C. Cardinal).

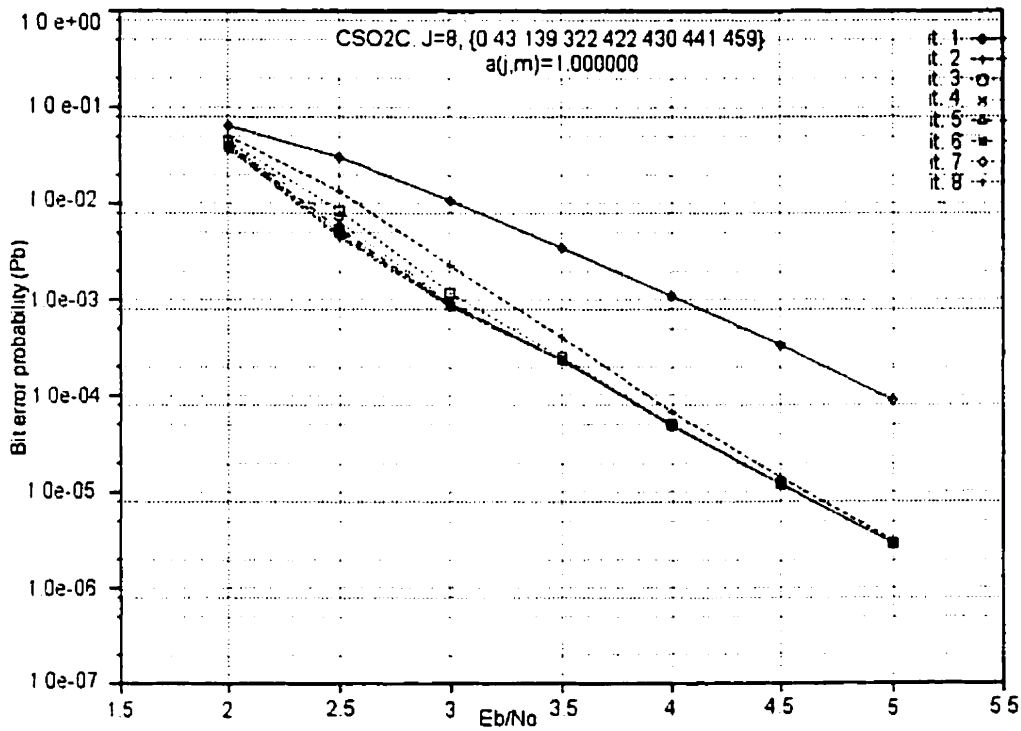

Figure 4.5 - *Probabilité d'erreur par bit en fonction de* $\frac{Eb}{No}$ *pour* $J = 8$ *(facteurs de gain* $a(j,m) = 1$*)*

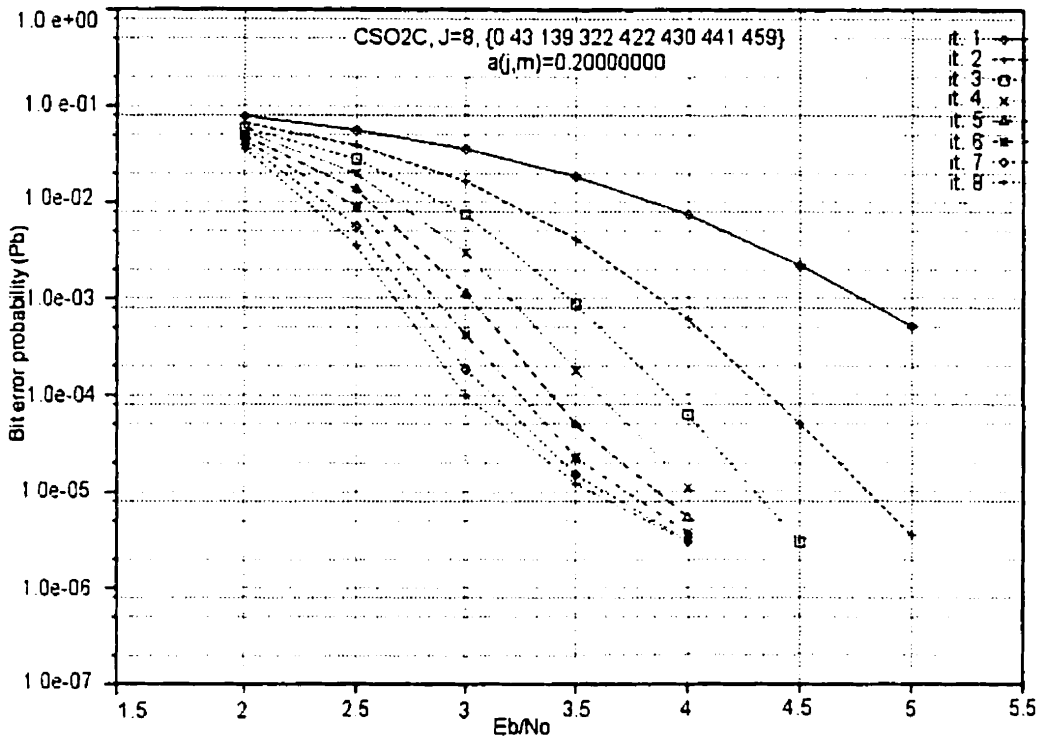

Figure 4.6 – *Probabilité d'erreur par bit en fonction de* $\frac{Eb}{No}$ *pour* $J = 8$ *(facteurs de gain "optimisés"* $a(j,m) = 0.2$*)*

Ces deux courbes font apparaître deux formes bien distinctes. L'influence du nombre d'itérations apparaît très nettement dans le cas où les gains ont été optimisés alors que les performances saturent au bout de trois itérations dans le cas où aucun réglage n'a été effectué. Ceci est tout à fait logique puisque sans aucune optimisation la corrélation entre les différents signaux mis en jeu devient rapidement trop forte et empêche l'obtention de meilleures performances. Si les différences de performances demeurent modestes pour des rapports signal à bruit élevés, des gains importants sont obtenus aux alentours d'un $\frac{Eb}{No}$ égal à 3. Cependant, il convient de rester vigilant sur le fait qu'aboutir à des différences de performances significatives nécessite un nombre d'itérations élevé (≥ 7) ce qui présente de nets inconvénients quant au temps de traitement. Pour illustrer les propos précédents, on présente dans le tableau 4.15 quelques valeurs significatives.

Tableau 4.15 – *Influence du réglage des gains d'un code d.o. au sens large de taux* $\frac{1}{2}$ *comportant 8 générateurs*

$\frac{Eb}{No}$	nb. itérations	Prob. d'erreur par bit	
		gains non opt.	gains opt.
3	8	10^{-3}	10^{-4}
4	8	$0.5\ 10^{-5}$	$0.8\ 10^{-4}$

On se référera à l'étude de C. Cardinal [9] pour de plus amples informations.

Il est également très intéressant d'estimer "l'effort de réduction" que notre méthode nécessite. Pour cela, il est possible de représenter la longueur de l'ensemble obtenu en fonction du temps et du nombre d'éléments qu'il contient. L'approche utilisée est la suivante : on lance la simulation avec un nombre d'éléments fixé et une limite de temps au bout de laquelle on l'arrête (ex : pour $J = 6$, la valeur d'1h30 a été choisie). On effectue de la sorte 10 essais. On obtient alors pour chaque tentative le temps qu'il a fallu pour atteindre les différentes longueurs au cours de la génération. Il est à noter que l'on obtient de grandes fluctuations d'une simulation à l'autre (ex : $J = 6$, temps mis pour aboutir à 100 variant de 48s à 1h10). On décide alors de considérer une moyenne tout en gardant à l'esprit la remarque précédente.

On présente dans le tableau 4.16 le type de valeurs obtenues après 10 simulations pour $J = 5$. La représentation en moyenne correspondante est reportée dans le tableau 4.17 (la valeur entre parenthèse inscrite dans le titre indique la longueur minimale obtenue pour l'ensemble considéré). Les tableaux concernant les valeurs de J comprises entre 6 et 10 figurent à l'Annexe IV (par convention ' représentera les minutes et " les secondes).

$J = 5$ (limite de temps: 1h)

Tableau 4.16 – *Longueurs obtenues et temps nécessaire lors de 10 tentatives pour* $J = 5$

88	*inst*	50	45"	47	2'05"	46	4'55"	43	8'55"	42	16'25"	41
77	*inst*	44	7"	43	11'50"	42	30'	41				
79	*inst*	48	10"	43	45"	42	19'30"	41				
91	*inst*	47	1'10"	43	3'05"	42	3'23"	41				
74	*inst*	46	1'10"	43	5'15"	42	11'40"	41				
105	*inst*	50	13"	45	26"	44	1'01"	43	3'35"	42	17'50"	41
65	*inst*	51	15"	43	28'	42	33'	41				
78	*inst*	51	8"	48	26"	42	6'16"	41				
68	*inst*	49	24"	47	1'20"	46	3'58"	41				
68	*inst*	46	3'10"	45	3'18"	44	4'20"	43	8'25"	42	15'15"	41

$J = 5$ (limite de temps: 1h)

Tableau 4.17 – *Évolution de la longueur moyenne en fonction du temps pour* $J = 5$ *(41)*

longueur	50	46	45	43	42	41
temps moyen	inst.	$\leq$ 10"	$\leq$ 43"	$\leq$ 5'	$\leq$ 30'	$\leq$ 45'

La figure 4.7 représente sous forme "d'histogramme" la fourchette de temps nécessaire à l'obtention de différentes longueurs en fonction du nombre de générateurs. Il est à noter que, comme cela fut le cas pour les tableaux précédemment présentés, la valeur minimale atteinte n'apparaît que rarement dans les grandeurs considérées. Afin de présenter des caractéristiques significatives, les résultats pris en compte proviennent de moyennes sur diverses tentatives. Il ne serait donc pas cohérent d'y insérer des valeurs n'ayant pour la plupart été atteinte qu'une unique fois.

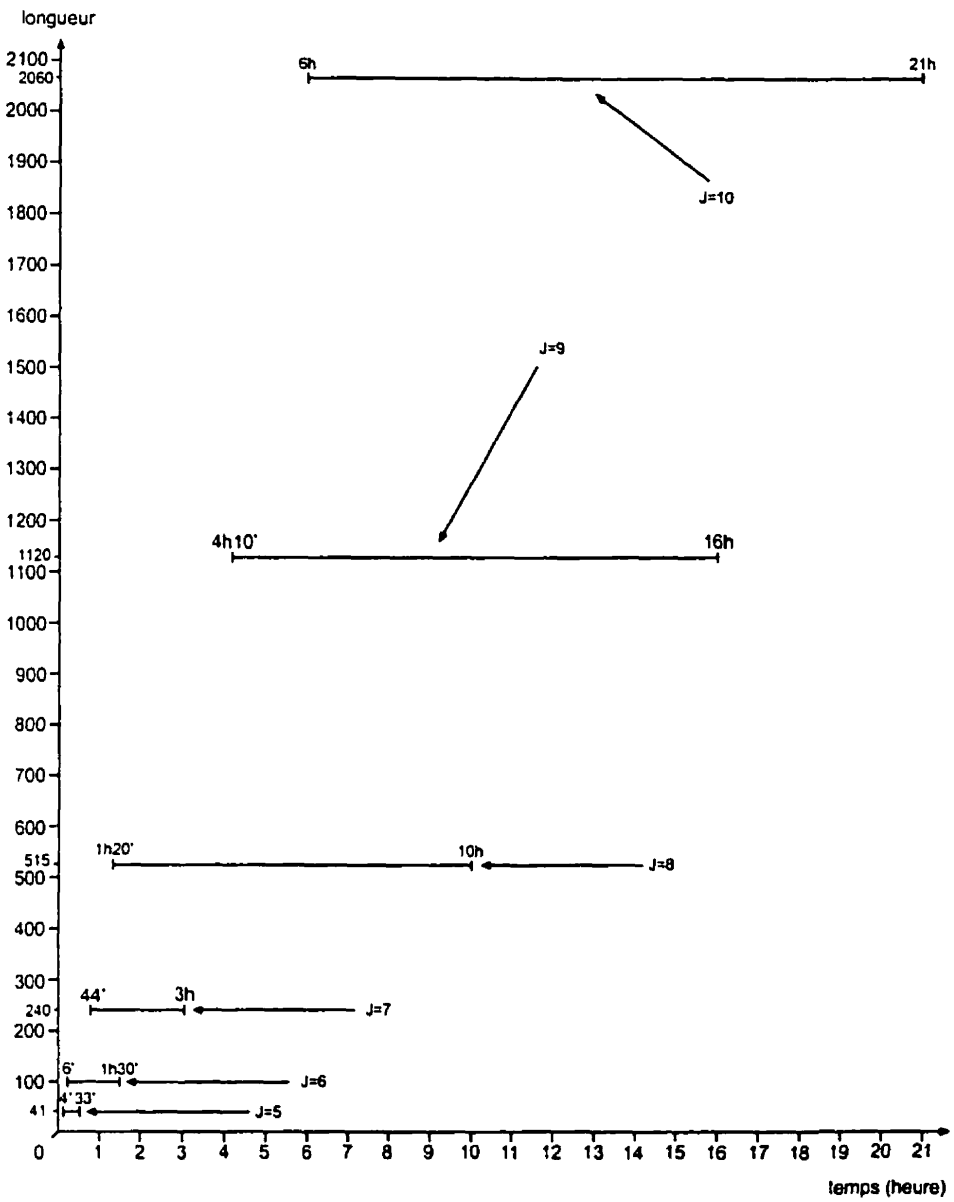

Figure 4.7 – *"Histogramme" des temps moyens obtenus dans le cas des codes d.o. au sens large*

Un simple fait illustrant parfaitement la limitation de nos méthodes vis-à-vis de l'augmentation du nombre d'éléments concerne le temps nécessaire à l'obtention d'une longueur de 10% supérieure à celle minimale obtenue.

Tableau 4.18 – *Temps nécessaire pour arriver à 10% de la longueur minimale obtenue dans les cas $J = 5$ et $J = 10$*

nb. d'éléments	long. minimale obtenue	10% de plus	temps nécessaire
5	41	45	$\approx$ 40"
10	1698	1868	$\approx$ 48h

Il est intéressant, maintenant que la phase de réduction a été effectuée, de comparer les nouveaux codes d.o. obtenus avec ceux précédemment connus et générés par l'intermédiaire de la géométrie projective avant une réduction basée sur une méthode de modulo [13].

Tableau 4.19 – *Comparaisons des longueurs minimales pour les ensembles complets initiaux et ceux obtenus durant notre étude ($J \leq 10$)*

J	initialement	générateurs obtenus	gain (%)
2	1	1	0
3	5	5	0
4	15	15	0
5	63	41	35
6	180	100	45
7	446	222	51
8	745	459	39
9	1500	912	40
10	2457	1698	31

On remarque que les améliorations apportées au niveau de la longueur minimale sont significatives ($\geq$ 30%). Du fait du caractère aléatoire des méthodes mises en jeu, la variation du gain en fonction de J n'est pas "linéaire".

On peut également revenir sur l'étude de la borne en comparant les longueurs générées pour les ensembles sans les négatifs (tableau 4.20) et les ensembles complets (tableau 4.21).

Tableau 4.20 – *Comparaisons longueurs obtenues - bornes établies pour les ensembles sans les négatifs*

J	longueur obtenue	borne cas idéal ($B_{inf}(J)$)	récursive ($B_{sup}(J)$)
2	1	1	-
3	4	4	-
4	12	11	12
5	33	25	37
6	79	50	112
7	174	91	337
8	375	154	1012
9	913	246	3037
10	1698	375	9112

Tableau 4.21 – *Comparaisons longueurs obtenues - bornes établies pour les ensembles complets*

J	longueur obtenue	borne cas idéal ($B_{inf}(J)$)	entiers manquants ($B_{appr}(J)$)	sommes ($B_{inf1}(J)$)
4	15	13.5	15	6
5	41	32.5	41	30
6	100	67.5	100	90
7	222	126	216	210
8	459	217	420	420
9	912	351	750	756
10	1698	540	1251	1260
11	3490	797.5	1975	1980
12	5173	1138.5	2981	2970
13	10201	1579.5	4335	4290
14	16285	2138.5	6110	6006
15	29532	2835	8386	8190

Il est difficile de parler de façon absolue car les longueurs obtenues présentées dans les deux tableaux précédents ne sont pas exactes pour $J \geq 7$. Il existe donc forcément un décalage entre la borne idéale et les valeurs présentées. On remarque cependant l'importante différence obtenue lorsque l'on compare avec la limite du cas idéal. Certaines sont cependant relativement fidèles pour de faibles valeurs de J.

Pour les deux types d'ensemble, on remarque que la longueur minimale obtenue progresse approximativement avec J selon 2^J. Ce fait est important puisqu'a priori la génération par géométrie projective assure une évolution de la longueur obtenue selon J^4 [13]. Ainsi, il est plus que probable que pour des valeurs élevées de J, les méthodes pseudo-aléatoires deviennent moins efficaces (en terme de longueur minimale) que la génération par géométrie projective. Cependant il ne faut pas oublier que les valeurs trop grandes de J conduisent à des longueurs de code inexploitables en pratique et que le temps demandé, pour produire par géométrie projective des codes d.o. contenant beaucoup d'éléments est énorme.

4.6 Conclusion

Les différentes méthodes de génération et de réduction ont permis de trouver des codes doublement orthogonaux dont la longueur est peu élevée. Les simulations de ces générateurs ont fait apparaître le très bon niveau de performances atteint. Même s'il reste possible d'améliorer la longueur minimale obtenue pour peu que l'on dispose de beaucoup de temps et d'une forte puissance de calcul les ensembles obtenus présentent à ce niveau une très nette amélioration par rapport aux anciens connus. Cette amélioration permet de réduire la mémoire du code et la latence du système de décodage et c'est là qu'elle revêt toute son importance.

CHAPITRE 5

CODES DOUBLEMENT ORTHOGONAUX DE TAUX 1/2 AU SENS STRICT

5.1 Introduction

On a vu qu'il était impossible d'obtenir des codes de taux $\frac{1}{2}$ strictement orthogonaux à cause des permutations inévitables (3.4.2.3). Il serait cependant intéressant de pouvoir obtenir des codes d.o. au sens strict qui présenteraient a priori de meilleures performances. On serait ainsi assuré qu'une inversion récursive du même ensemble de parité n'utiliserait pas le même ensemble d'observables. Cette indépendance entraînerait certainement l'obtention de plus faibles probabilités d'erreur et aussi d'un plus petit nombre d'itérations. En fait, on va s'apercevoir qu'obtenir de tels codes est possible mais seulement pour des taux de codage du type $\frac{mK}{mN}$ (avec m entier positif).

Le but recherché est donc de limiter les permutations possibles au sein des équations de parité. Une idée simple introduite par F. Gagnon et D. Haccoun [12] pour réaliser cela est de faire correspondre chaque séquence d'information avec un et uniquement un symbole d'une séquence de parité donnée. En procédant de la sorte, le code obtenu sera de taux $\frac{J}{2J}$. Les générateurs seront mis sous la forme $g_{k,n}$ représentant par cette notation la "position" de la connexion entre la séquence de parité n et la séquence d'information k. La figure 5.1 représente un exemple illustratif dans le cas d'un code de taux $\frac{3}{6}$ systématique.

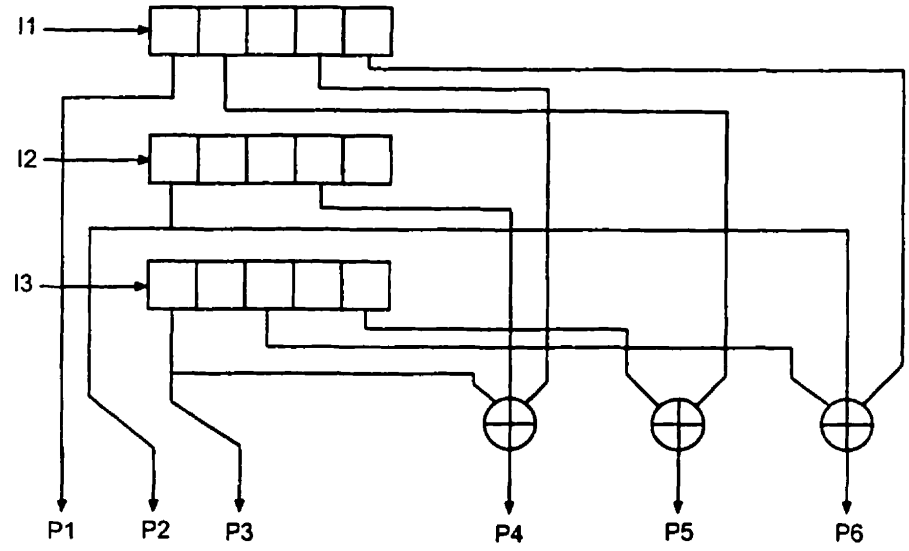

Figure 5.1 – *Exemple de structure d'un code systématique défini au sens strict de taux* $\frac{3}{6}$

L'expression des équations de parité donne :

$$P_i^k = \sum_{m=1}^{J} I_m^{k-g_{m-1,i-1}} \tag{5.1}$$

où l'on a :

- P_i^k : *symbole de la séquence de parité i à "l'instant" k*
- I_i^k : *symbole de la séquence d'information i à "l'instant" k*
- $g_{s,l}$: *générateur entre la séquence d'information s et la séquence de parité l*

Dans notre exemple, on a $g_{1,4} = 3$, $g_{2,4} = 3$ *et* $g_{3,4} = 0$ soit $P_4^k = I_1^{k-3} \oplus I_2^{k-3} \oplus I_3^{k-0}$. De manière générale, lorsqu'un système itératif de décodage est utilisé, on arrive au bout de deux itérations à utiliser les symboles reçus du type :

$$\hat{I}_v^{j-g_{k,l}+g_{m,l}-g_{m,n}+g_{v,n}}. \tag{5.2}$$

Ainsi pour toutes les paires (v,k) données, les expressions $g_{k,l} - g_{m,l} + g_{m,n} - g_{v,n}$ doivent être distinctes quel que soit (l,m,n) tel que $n \neq l$, $m \neq k$ et $m \neq v$ (ordre

2). En formalisant ceci sous une forme mathématique on obtient la proposition suivante :

L'ensemble de générateurs $g_{i,j}$ (notation matricielle) constitue un code doublement orthogonal au sens strict ssi

$$\begin{aligned} &\textit{Soit}(v,k),\ \textit{quels que soient}\ (l,m,n)\ \textit{et}\ (l',m',n')\ \textit{tels que}\ n \neq l,\ m \neq k\ \textit{et}\ m \neq v\ ; \\ &n' \neq l',\ m' \neq k'\ \textit{et}\ m' \neq v'\ ;\ (l,m,n) \neq (l',m',n')\ \textit{on a}: \\ &g_{k,l} - g_{m,l} + g_{m,n} - g_{v,n} \neq g_{k,l'} - g_{m',l'} + g_{m',n'} - g_{v,n'}. \end{aligned} \tag{5.3}$$

Pour expliciter la notation matricielle plus en détails, on présente l'exemple d'une matrice de générateurs d'un code de taux $\frac{2}{4}$ et la réalisation matérielle qui s'y rapporte (on utilise exclusivement des codes systématiques) :

Tableau 5.1 – *Matrice de générateurs d'un code de taux* $\frac{2}{4}$

$$\begin{pmatrix} 0 & 3 \\ 2 & 4 \end{pmatrix}$$

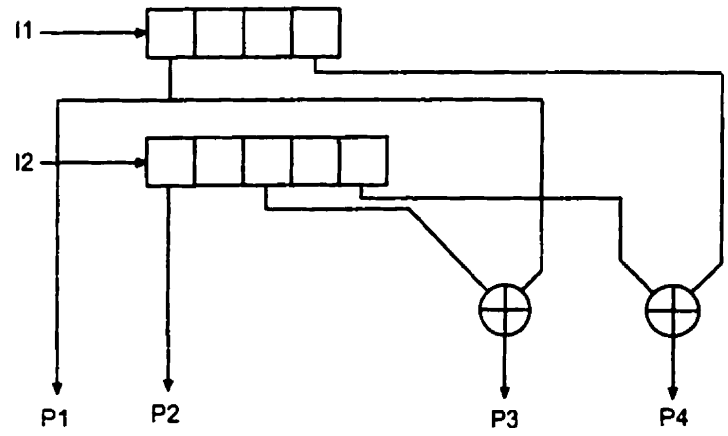

Figure 5.2 – *Réalisation matérielle du codeur de taux* $\frac{2}{4}$ *dont les générateurs sous forme matricielle figurent au tableau 5.1*

Notre but sera donc, identiquement à ce qui a été fait jusqu'à présent de générer des ensembles de ce type puis de réduire ceux-ci au maximum.

5.2 Génération

5.2.1 Géométrie projective

5.2.1.1 Principe

Une fois encore il est possible d'utiliser la géométrie projective. C'est d'ailleurs la technique initialement employée par F. Gagnon et D. Haccoun [12]. Le principe de cette méthode a déjà été explicité (4.2) et il demeure le même ici en se plaçant dans PG(4, p^s). Le choix de p et de s s'effectue selon le critère : $J^2 \leq p^s + 1$. Il existe alors de nombreuses façons de choisir les générateurs. Une solution retenue est :

$$\begin{cases} \alpha^{g_{0,0}} = 1 \\ \alpha^{g_{1,0}} = \alpha \\ \alpha^{g_{k,l}} = 1 + \beta^{(k+lJ-2)}.\alpha,\ (k+Jl) \geq 2. \end{cases} \tag{5.4}$$

L'expression de la condition de double orthogonalité devient (en permutant des termes pour l'exprimer sous forme de sommes) :

$$(1+\beta^{(k+l'.J-2)}.\alpha)(1+\beta^{(m+l.J-2)}.\alpha)(1+\beta^{(m'+n'.J-2)}.\alpha)(1+\beta^{(v+n.J-2)}.\alpha)$$
$$\neq$$
$$(1+\beta^{(k+l.J-2)}.\alpha)(1+\beta^{(m'+l'.J-2)}.\alpha)(1+\beta^{(m+n.J-2)}.\alpha)(1+\beta^{(v+n'.J-2)}.\alpha)$$

Le développement de cette expression fait apparaître de part et d'autre du signe "$\neq$" le même coefficient du terme en α^4. Il n'est donc pas utile de se placer

dans PG(4,p^s) et PG(3,p^s) est suffisant. La répartition des générateurs présentée n'est naturellement pas unique et d'autres ont été envisagées. Particulièrement on remarque que la technique employée consiste toujours en la projection sur une ligne ; il fut également essayé de projeter sur un plan ce qui ne donna pas lieu à de meilleurs résultats.

5.2.1.2 Résultats obtenus

Les résultats présentés ici sont les meilleurs générés par cette méthode. Une procédure adaptée à partir du programme *Maple*© réalisé au 4.2.1.3 a été utilisée. Malheureusement, le nombre de points à calculer étant cette fois-ci J^2 les calculs deviennent extrêmement longs dès que J augmente. Il n'a d'ailleurs pas été possible d'aller au delà de la valeur $J = 6$ (ce qui était d'ailleurs semble-t-il le cas pour les générations antérieures [12])

Tableau 5.2 – *Générateurs de codes d.o. au sens strict obtenus par géométrie projective (notations identiques à celles du chapitre précédent)*

J	générateurs	paramètres		
		p	s	modulo
2	$\begin{pmatrix} 0 & 4 \\ 1 & 13 \end{pmatrix}$	3	1	40
3	$\begin{pmatrix} 0 & 328 & 84 \\ 1 & 89 & 37 \\ 404 & 558 & 490 \end{pmatrix}$	2	3	585
4	$\begin{pmatrix} 0 & 3910 & 2017 & 3481 \\ 1 & 586 & 1539 & 4166 \\ 315 & 1805 & 2876 & 1011 \\ 3770 & 4328 & 2152 & 351 \end{pmatrix}$	2	4	4389
5	$\begin{pmatrix} 0 & 1006 & 3646 & 11350 & 1799 \\ 1 & 12157 & 12310 & 3331 & 8530 \\ 821 & 15244 & 229 & 4213 & 3588 \\ 3046 & 2135 & 2947 & 15066 & 6275 \\ 9628 & 3698 & 13850 & 1002 & 4919 \end{pmatrix}$	5	2	16276

On verra par la suite que la longueur de ces ensembles est bien loin de celle qui sera atteinte après réduction.

5.2.2 Génération pseudo-aléatoire

5.2.2.1 Présentation

Cette technique est le prolongement aux codes d.o. au sens strict de celle vue au 4.2.2. Elle présente l'avantage d'être particulièrement rapide et de donner de forts honorables résultats même s'ils présentent certaines particularités.
Son fonctionnement est le suivant :

1. On part d'un ensemble de générateurs doublement orthogonaux au sens strict obtenu par la méthode précédente ou par toute autre méthode (exemple : $g_{i,j} = 10^{i*J+j}$). On définit une variable "vargen" qui s'identifie initialement au générateur $g_{0,0}$ et une variable entière "compteur".
2. On initialise "compteur" à 0.
3. On incrémente "compteur".
4. On remplace "vargen" par "compteur". On vérifie si l'ensemble ainsi généré est doublement orthogonal. Si non, on retourne à l'étape 3. Si oui on effectue un test à caractère aléatoire. Si ce test est validé, on conserve le générateur obtenu et "vargen" s'identifie maintenant à l'élément suivant (ordre naturel) de notre matrice de générateurs, on reprend alors à l'étape 2. Si le test échoue, on reprend à l'étape 3 sans rien modifier.
5. On continue de la sorte jusqu'à obtenir le nombre de générateurs recherché.

Le test aléatoire est une nouvelle fois la partie délicate de cette stratégie. Après plusieurs essais il s'est avéré que l'on a retrouvé des comportements similaires à ceux qui ont été exposés au 4.2.2.1. Les mêmes tests ont donc été appliqués ici.
L'utilisation de cette technique ne demande pas un grand temps de calcul et il a été possible de considérer les générateurs de codes de taux $\frac{10}{20}$.

5.2.2.2 Résultats

A titre de comparaison avec la géométrie projective [13], on présente dans le tableau 5.3 les résultats obtenus pour $J \leq 5$. La différence au niveau des longueurs est plus que significative:

Tableau 5.3 – *Comparaisons des techniques de géométrie projective et pseudo-aléatoire pour des codes de taux* $\frac{J}{2J}$ *avec* $J \geq 5$

J	gén. pseudo-aléa.	longueur	longueur géo. proj.
2	$\begin{pmatrix} 0 & 0 \\ 11 & 13 \end{pmatrix}$	13	13
3	$\begin{pmatrix} 6 & 12 & 6 \\ 1 & 3 & 0 \\ 6 & 1 & 19 \end{pmatrix}$	19	558
4	$\begin{pmatrix} 10 & 3 & 31 & 6 \\ 4 & 0 & 0 & 8 \\ 25 & 3 & 1 & 64 \\ 13 & 22 & 61 & 133 \end{pmatrix}$	133	4328
5	$\begin{pmatrix} 0 & 13 & 6 & 6 & 11 \\ 9 & 5 & 10 & 1 & 31 \\ 13 & 2 & 87 & 18 & 98 \\ 20 & 60 & 102 & 155 & 0 \\ 20 & 54 & 191 & 233 & 12 \end{pmatrix}$	233	15244

Là encore (section 4.5), la génération pseudo-aléatoire contient une phase de réduction intrinsèque de par le fait qu'un choix est effectué entre divers ensembles générés. La comparaison est donc à prendre avec un certain recul. Cependant si l'on appliquait d'ores et déjà la phase de réduction complète (section 5.3) aux ensembles obtenus par géométrie projective la différence demeurerait significative (tableau 5.4).

Tableau 5.4 – *Comparaisons des techniques de géométrie projective après réduction et pseudo-aléatoire pour des codes de taux* $\frac{J}{2J}$ *avec* $J \geq 5$

J	gén. pseudo-aléa.	géo. proj. (après réduc.)
2	$\begin{pmatrix} 0 & 0 \\ 11 & \underline{13} \end{pmatrix}$	$\begin{pmatrix} 0 & 4 \\ \underline{4} & 0 \end{pmatrix}$
3	$\begin{pmatrix} 6 & 12 & 6 \\ 1 & 3 & 0 \\ 6 & 1 & \underline{19} \end{pmatrix}$	$\begin{pmatrix} 0 & \underline{120} & 54 \\ 120 & 0 & 0 \\ 0 & 72 & 56 \end{pmatrix}$
4	$\begin{pmatrix} 10 & 3 & 31 & 6 \\ 4 & 0 & 0 & 8 \\ 25 & 3 & 1 & 64 \\ 13 & 22 & 61 & \underline{133} \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 & 3792 \\ 3325 & 0 & 905 & 3765 \\ 479 & 0 & 1023 & 636 \\ 3108 & \underline{3792} & 323 & 0 \end{pmatrix}$
5	$\begin{pmatrix} 0 & 13 & 6 & 6 & 11 \\ 9 & 5 & 10 & 1 & 31 \\ 13 & 2 & 87 & 18 & 98 \\ 20 & 60 & 102 & 155 & 0 \\ 20 & 54 & 191 & \underline{233} & 12 \end{pmatrix}$	$\begin{pmatrix} 491 & 0 & 175 & 1911 & 8959 \\ 497 & 11156 & \underline{13598} & 0 & 2029 \\ 4554 & 12726 & 0 & 2229 & 13598 \\ 11508 & 2997 & 3234 & 13598 & 0 \\ 0 & 6239 & 652 & 2850 & 1960 \end{pmatrix}$

5.2.3 Répartition aléatoire des indices

5.2.3.1 Présentation

Le principe repose ici aussi sur un choix aléatoire. Le cheminement cependant un peu différent est présenté ci dessous (on appelle M le nombre de générateurs de l'ensemble) :

1. On choisit comme état initial un ensemble de générateurs doublement orthogonaux au sens strict (exemple : $g_{i,j} = 10^{i*J+j}$).
2. On considère les M premiers entiers (0 inclus). On les ordonne sous forme de liste de manière aléatoire. Chacun des entiers n précédents correspond au générateur situé à la $n^{ième}$ position de la matrice $(g_{i,j})$ contenant tous les générateurs $(n = i * J + j)$.
3. On considère le générateur $g_{i,j}$ correspondant au premier indice de la liste

obtenue aléatoirement.

4. On fait prendre à ce générateur les valeurs des entiers naturels dans l'ordre croissant en commençant par zéro. Dès qu'un ensemble doublement orthogonal est obtenu, on conserve cette valeur pour ce générateur et on considère le générateur correspondant à l'indice suivant de la liste. On recommence alors cette étape jusqu'à avoir rempli ainsi notre matrice d'éléments.

La méthode ainsi implémentée n'ayant qu'un seul test aléatoire à effectuer durant toute la procédure est extrêmement rapide. Elle donne de bons résultats (ordres de grandeur similaires à ceux obtenus par une génération pseudo-aléatoire).

5.2.4 Variante (répartition aléatoire des indices n^o2)

Une évolution efficace de cette méthode a été obtenue. Celle ci consiste à utiliser la répartition aléatoire des indices de manière différente. Le cheminement est le suivant :

1. On part d'une matrice de générateurs doublement orthogonaux.
2. On repartit les M premiers entiers sous forme de liste aléatoire L de la même façon que précédemment.
3. On considère une variable V initialement égale à 0.
4. On tente de remplacer chaque générateur de notre matrice par la valeur V suivant l'ordre défini par L de sorte à conserver une matrice doublement orthogonale.
5. On incrémente V et on réitère l'étape précédente jusqu'à ce que la matrice soit complètement remplie.

5.2.5 Comparaison des résultats

On a déjà pu avoir une idée de la différence au niveau de la longueur obtenue que pouvait entraîner l'utilisation d'une méthode à caractère aléatoire vis à vis

d'une technique comme la géométrie projective qui repose sur une mise en forme algébrique. Le gain en vitesse d'exécution est également très visible et c'est là un point très important. N'oublions cependant pas que c'est après réduction que ces ensembles nous intéressent et qu'il serait hâtif de tirer des conclusions définitives à ce stade de l'étude.

5.3 Réduction

5.3.1 Introduction

La problématique demeure la même, il s'agit de réduire au maximum la longueur des codes générés dans le but de limiter les besoins en mémoire et la latence au niveau du traitement. La forme matricielle sous laquelle on a décidé de répartir les générateurs est fort commode pour visualiser la condition de double orthogonalité au sens strict. Illustrons cela par une simple matrice de taille 5 par 5.

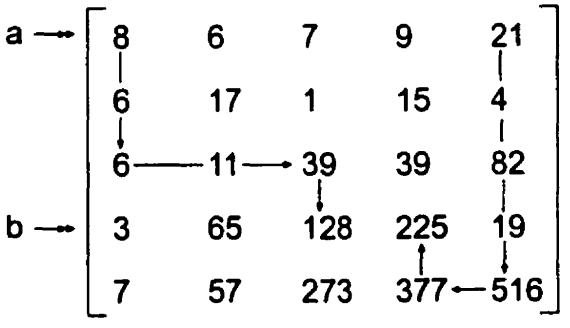

Figure 5.3 – *Illustration de la condition de double orthogonalité au sens strict*

L'équation 5.3 signifie que tous les trajets partant d'un élément d'une ligne a et se rendant à une ligne b en suivant l'évolution suivante : déplacement "vertical" puis "horizontal" et enfin "vertical" doivent être différents. Ceci représente un certain nombre d'expressions qui doivent être distinctes. On peut les dénombrer de

manière relativement simple :

- Partant ligne i, arrivant ligne j $(i \neq j)$

$$\underbrace{\binom{J}{1}}_{\substack{\text{choix du premier} \\ \text{élément ligne i}}} \cdot \underbrace{\binom{J-2}{1}}_{\substack{\text{choix du deuxième élément} \\ \text{– même colonne qu'auparavant} \\ \text{mais élément différent} \\ \text{– ligne différente} \\ \text{de celle d'arrivée}}} \cdot \underbrace{\binom{J-1}{1}}_{\substack{\text{choix du troisième élément} \\ \text{sur la même ligne que le} \\ \text{précédent mais différent} \\ \text{de celui–ci}}} \cdot \underbrace{1}_{\substack{\text{choix du dernier} \\ \text{élément colonne} \\ \text{et ligne imposées}}}$$

Ce qui représente $J.(J-1).(J-2)$ équations

- Partant ligne i, retournant ligne i

La démarche suivie est la même que précédemment . Le choix du deuxième élément, du fait qu'il soit distinct du premier assure qu'il ne sera pas sur la ligne d'arrivée. On obtient ainsi :

$$\binom{J}{1} \cdot \binom{J-1}{1} \cdot \binom{J-1}{1} .1$$

Soit $J.(J-1)^2$ possibilités.

5.3.2 Théorie

La première idée est de réutiliser les concepts étudiés lors du chapitre précédent. On pense notamment à la technique ayant recours à des opérations élémentaires modulo un entier. Celle ci est en effet facilement applicable ici. Pour fixer les idées on peut imaginer insérer tous les générateurs de notre matrice dans un vecteur ligne et considérer cet ensemble comme un code doublement orthogonal au sens large. Il

a cependant déjà été évoqué que le temps de calcul augmentait fortement avec le nombre d'éléments. Dans notre cas on considère un nombre de générateurs de la forme J^2. Dans ces conditions, une telle technique s'avère difficilement exploitable.

La recherche exhaustive se heurte également au même problème et apparaît même avec une puissance de calcul importante inenvisageable.

La méthode qui a été utilisée se base sur les recherches effectuées par W. W. Wu [46] dans le cas des codes simplement orthogonaux. Partant d'une matrice de générateurs initiale (obtenue par une des méthodes du point précédent) et en effectuant de simples opérations d'addition et de soustraction selon un ordre judicieusement choisi, on aboutit à une situation d'arrêt qu'aucune tentative utilisant le même type d'opérations ne saurait réduire.

On peut formaliser la procédure ainsi :

$$\underbrace{M}_{\text{matrice de générateurs d.o. initiale}} = \underbrace{M'}_{\text{matrice d.o. réduite}} + \underbrace{H}_{\text{matrice "ligne"}} + \underbrace{V}_{\text{matrice "colonne"}}$$

où

$$H = \begin{pmatrix} w & x & y & \dots & z \\ w & x & y & \dots & z \\ w & x & y & \dots & z \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix} \text{ et } V = \begin{pmatrix} W & W & W & \dots & W \\ X & X & X & \dots & X \\ Y & Y & Y & \dots & Y \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix}$$

Ces formes particulières proviennent du type d'opérations utilisées.

La situation d'arrêt correspond à l'obtention au sein de la matrice M' d'un cycle où se succèdent alternativement des valeurs maximales (ou maximales-1) et des zéros en effectuant tour à tour des déplacements horizontaux et verticaux. Ce

propos est illustré par la matrice ci-après:

$$\begin{pmatrix} \underline{N} & \underline{0} & r \\ \underline{0} & s & \underline{N-1} \\ t & \underline{N} & \underline{0} \end{pmatrix} \text{ avec } 0 \leq r,s,t < N$$

Pour s'assurer que l'on ne pourra pas faire mieux, il est possible d'effectuer la démonstration suivante (on considérera à titre d'exemple le cas de notre matrice 3*3 présentée ci avant).

On raisonne par l'absurde en supposant l'existence d'une autre matrice M'' dont le plus grand élément est inférieur au plus grand élément de M. On a alors:

$$M' = M'' + H + V$$

Soit:

$$\begin{pmatrix} N & 0 & r \\ 0 & s & N-1 \\ t & N & 0 \end{pmatrix} = M'' + \begin{pmatrix} a & a & a \\ b & b & b \\ c & c & c \end{pmatrix} + \begin{pmatrix} A & B & C \\ A & B & C \\ A & B & C \end{pmatrix}$$

D'après l'hypothèse formulée sur M'' on a:

$$max(M'') - min(M'') < max(M') - min(M').$$

En reformulant l'équation précédemment établie, on aboutit à:

$$M'' = \begin{pmatrix} N & 0 & r \\ 0 & s & N-1 \\ t & N & 0 \end{pmatrix} - \begin{pmatrix} a & a & a \\ b & b & b \\ c & c & c \end{pmatrix} - \begin{pmatrix} A & B & C \\ A & B & C \\ A & B & C \end{pmatrix}$$

D'où:

$$M'' = \begin{pmatrix} N-a-A & -a-B & r-a-C \\ -b-A & s-b-B & N-1-b-C \\ t-c-A & N-c-B & -c-C \end{pmatrix}$$

On en tire les inégalités suivantes :

$$\begin{cases} max(M'') \geq N - a - A \\ min(M'') \leq -b - A \end{cases} \Rightarrow$$

$$N - a - A - (-b - A) \leq max(M'') - min(M'') < max(M') - min(M')$$

$$\Rightarrow \underline{N - a + b < max(M') - min(M')}$$

de même

$$N - c - B - (-a - B) < max(M') - min(M')$$

$$\Rightarrow \underline{N - c + a < max(M') - min(M')}$$

et

$$N - 1 - b - C - (-c - C) < max(M') - min(M')$$

$$\Rightarrow \underline{N - 1 - b + c < max(M') - min(M')}$$

On arrive alors à :

$$\begin{cases} b - a < max(M') - min(M') - N \\ a - c < max(M') - min(M') - N \\ c - b \leq max(M') - min(M') - N \end{cases}$$

Avec $max(M') - min(M') = N$ on obtient :

$$\begin{cases} b - a < 0 \\ a - c < 0 \\ c - b \leq 0 \end{cases} \Rightarrow \begin{cases} b < a \\ a < c \\ c \leq b \end{cases} \Rightarrow b < a < c \leq b \Rightarrow b < b \ \ ABSURDE$$

Ainsi la présence d'un cycle de la forme décrite auparavant assure d'avoir abouti à une matrice "non-réductible" par la même méthode.

5.3.3 Stratégie employée

La stratégie employée combine une génération à répartition d'indices pseudo-aléatoire et la technique de réduction proposée. Pour garantir l'arrivée à une situation d'arrêt l'algorithme suivant a été mis en place :

- On considère tout d'abord les lignes de notre matrice.

- On repère le minimum de chaque ligne l. On soustrait alors ce nombre (s'il est différent de 0) à toute la ligne l.
- On fait de même sur les colonnes.
- On repère le (un des) maximum(s) absolu(s) de la matrice que l'on note $m\{i,j\}$ (i: ligne, j: colonne). On note n_1, n_2, ..., n_k les colonnes contenant un "0" sur la ligne i.
- On note m^* le maximum absolu contenu sur ces colonnes.
- Si $m^* \geq m\{i,j\} - 1$, on recommence la procédure précédente en notant m_1,...,m_k les lignes contenant un "0" sur la colonne j.
- On étudie m_1, m_2, ..., m_k pour y repérer le maximum n^*. (a)
  - Si $n^* \geq m\{i,j\} - 1$, la matrice obtenue est celle recherchée (situation d'arrêt).
  - Sinon on effectue l'étape suivante. (b)
- On calcule $d = \lfloor \frac{m\{i,j\} - m^*}{2} \rfloor$. On additionne d à toutes les colonnes n_1, ..., n_k si l'on vient de (a), à toutes les lignes m_1, ..., m_k si l'on vient de (b).
- On réitère la procédure avec la nouvelle matrice obtenue (on la transpose si l'on vient de (b)).

La procédure présentée a donné de grandes satisfactions au niveau des résultats et du temps d'exécution. Cependant pour $J^2 \geq 100$ le temps requis commence à être important.

5.3.4 "Optimisation"

Pour pallier à l'inconvénient précédent et espérer ainsi générer des codes de taux plus élevés, une évolution de la méthode précédente fut adoptée.

Celle-ci consiste en l'utilisation de la matrice de générateurs doublement orthogonaux de taille J^2 pour obtenir celle de taille $(J+1)^2$. En procédant de

la sorte, la répartition aléatoire d'indices ne concerne plus $(J+1)^2$ indices mais $(J+1)^2 - J^2 = 2.J + 1$ ce qui réduit d'autant le temps de calcul.

Le problème fut de savoir si agir de la sorte en "fixant" à priori une partie des générateurs ne restreignait pas les possibilités de réduire de façon intéressante la matrice ainsi obtenue. Diverses simulations réalisées dans le cas de $J = 5$ ont établi que l'on obtenait avec cette technique la même longueur que dans le cas précédent à condition de rajouter un nouveau paramètre aléatoire (p.a.) explicité au sein de la description de la procédure présentée ci-après.

- On part d'une matrice de générateurs doublement orthogonaux de taille 3*3.

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

- On additionne à chacun de ses générateurs un même nombre aléatoire x (p.a.).

$$\begin{pmatrix} a+x & b+x & c+x \\ d+x & e+x & f+x \\ g+x & h+x & i+x \end{pmatrix}$$

- On insère cette matrice dans celle d'ordre 4*4 recherchée.

$$\begin{pmatrix} a+x & b+x & c+x & A \\ d+x & e+x & f+x & B \\ g+x & h+x & i+x & C \\ D & E & F & G \end{pmatrix}$$

- On effectue alors une méthode de répartition aléatoire des indices (section 5.2.3) mais uniquement sur A, B, C, D, E, F et G.

5.3.5 Résultats obtenus

Par rapport aux codes initialement utilisés [13] (auxquels on a appliqué la méthode de réduction précédemment définie (5.3.3)) le gain au niveau de la longueur est substantiel. On peut même noter une réduction d'un facteur 60 dans le tableau 5.5.

Tableau 5.5 - *Comparaison des longueurs trouvées pour les codes au sens strict*

J	initialement	notre étude	amélioration(rapport avant/après)
2	1	1	1
3	62	5	12.4
4	957	25	38.28
5	6069	98	61.93
6	-	323	-
7	-	879	-
8	-	2252	-
9	-	4804	-
10	-	9778	-
11	-	18306	-
12	-	33161	-
13	-	52000	-
14	-	80901	-

5.4 Générateurs obtenus

Les tableaux présentés à l'Annexe V répertorient sous forme matricielle les générateurs ayant la plus faible longueur atteinte. Dans chaque cas, les générateurs formant le cycle d'arrêt sont soulignés.

Naturellement ces ensembles ne sont pas uniques. Une fois encore, rien ne garantit l'obtention de l'ensemble le plus minimal possible pour J fixé.

Au niveau temporel, la comparaison du temps nécessaire à l'obtention d'une matrice d.o. par les méthodes employées pour deux valeurs de J distinctes permet de se rendre compte de la limitation de nos techniques vis à vis de l'augmentation du nombre d'éléments:

Tableau 5.6 - *Temps nécessaire à l'obtention d'une matrice d.o. au sens strict de taux* $\frac{5}{10}$ *et* $\frac{14}{28}$

taux	temps pour créer une matrice d.o.
$\frac{5}{10}$	1"
$\frac{14}{28}$	270h

Du fait de l'utilisation d'un paramètre aléatoire, plus le nombre de matrices générées est grand et plus les chances d'obtenir un "bon résultat" augmentent. Ainsi si l'on peut obtenir 10000 codes distincts de taux $\frac{5}{10}$ en un peu moins de 2h50, il faudrait environ 308 ans dans les mêmes conditions pour en obtenir autant de taux $\frac{14}{28}$. De par ce fait les longueurs obtenues pour des codes de faibles taux sont sans nul doute plus près de "l'optimale" que celles obtenues pour des taux plus importants.

5.5 Simulation des codes générés

Les simulations qui ont été effectuées sur ces codes ont permis de mettre en évidence certaines particularités notamment vis à vis de leurs homologues au sens large. Ces caractéristiques seront présentées en détails dans la section suivante intitulée "comparaisons".

On remarque une nouvelle fois que la longueur du code n'influence pas les performances d'erreur de celui ci. L'explication intuitive présentée au (4.5) demeure la même. L'illustration de ce propos est réalisée par les deux courbes 5.4 et 5.5.

On s'aperçoit également qu'après la troisième itération, le gain en performances d'erreur à chaque nouvelle itération tend à devenir nul.

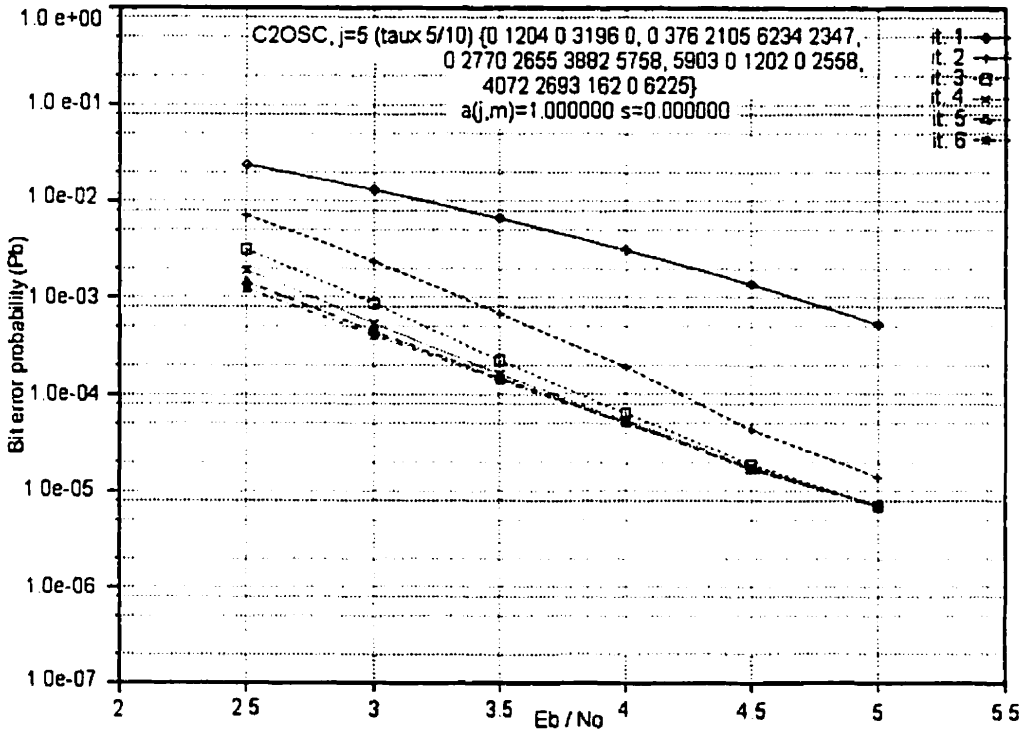

Figure 5.4 – *Probabilité d'erreur par bit en fonction de* $\frac{Eb}{No}$ *pour un code de taux* $\frac{5}{10}$ *d.o. au sens strict (longueur non réduite)*

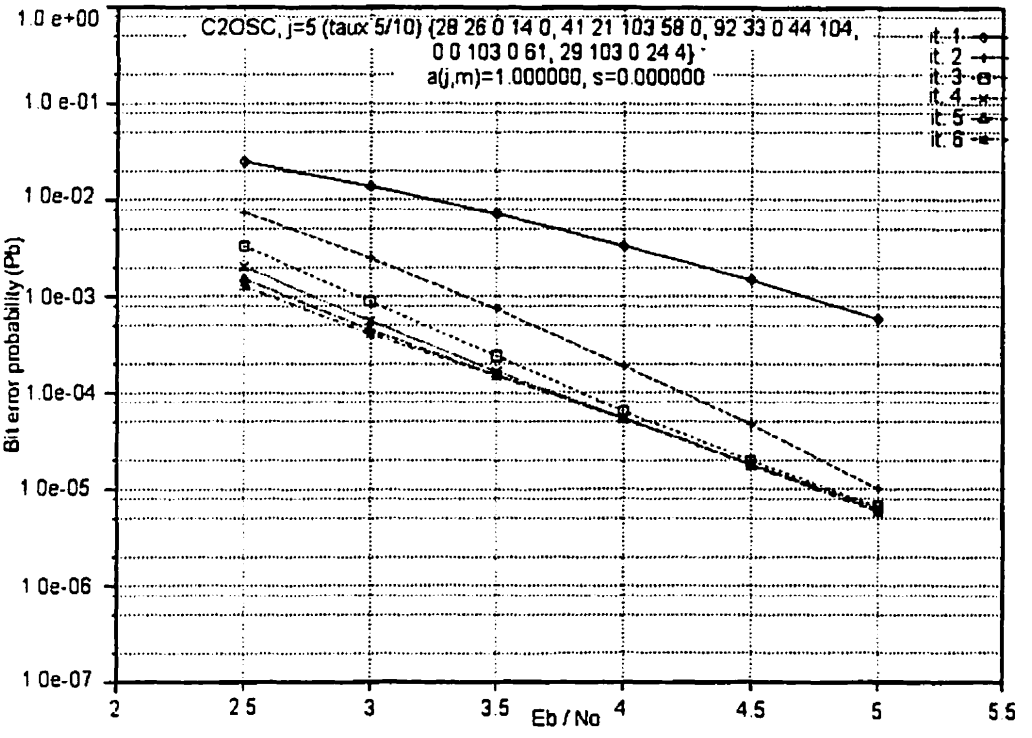

Figure 5.5 – *Probabilité d'erreur par bit en fonction de* $\frac{Eb}{No}$ *pour un code de taux* $\frac{5}{10}$ *d.o. au sens strict (longueur réduite)*

5.6 Comparaisons de la double orthogonalité au sens strict et au sens large

5.6.1 Longueurs des codes

Il apparaît intéressant maintenant que l'on a pu aborder le sens strict et le sens large de comparer numériquement les longueurs obtenues dans chacun des cas. Ainsi le tableau ci dessous compare la plus petite longueur atteinte lors de la génération d'un code d.o. au sens large de taux $\frac{1}{2}$ contenant J éléments à celle atteinte lors de la génération d'un code d.o. au sens strict de taux $\frac{J}{2J}$.

Tableau 5.7 - *Comparaison des longueurs minimales obtenues pour les deux types de codes doublement orthogonaux*

J	longueur minimale	
	sens large, complet, taux $\frac{1}{2}$, J générateurs	sens strict, taux $\frac{J}{2J}$
2	1	1
3	5	5
4	15	25
5	41	98
6	100	323
7	222	879
8	459	2252
9	912	4803
10	1698	9778
11	3490	18306
12	5173	33161
13	10201	52000
14	16285	80901

On s'aperçoit clairement en analysant le tableau 5.7 que la longueur minimale obtenue dans le cas des codes d.o. au sens strict devient très rapidement largement supérieure à celle atteinte dans le cas des codes d.o. au sens large.

5.6.2 Contraintes temporelles

Comme on l'a déjà évoqué, le nombre de générateurs dans le cas d'un code d.o. au sens strict de taux $\frac{J}{2J}$ est beaucoup plus élevé (J^2) que celui dans le cas d'un code d.o. au sens large de taux $\frac{1}{2}$ contenant J éléments (J). On s'attend donc logiquement à ce que le temps nécessaire dans le premier cas soit le plus élevé. Il faut cependant garder à l'esprit que les méthodes utilisées diffèrent. Ainsi si les deux techniques de génération font appel à une procédure pseudo-aléatoire et sont en ce sens relativement similaires, les techniques de réduction sont différentes et une procédure exacte (cas des codes d.o. au sens strict) s'oppose à une procédure plus "empirique" (cas des codes d.o. au sens large).

Du fait de l'utilisation de paramètres aléatoires il est difficile de comparer très précisément les deux types de codes sur un plan temporel. La stratégie adoptée a consisté à effectuer une moyenne sur le temps mis pour arriver à 10% de la meilleure longueur absolue obtenue dans chacun des cas (moyenne sur une dizaine d'essais). Les résultats sont les suivants :

Tableau 5.8 – *Comparaison temporelle à 10% de la longueur minimale obtenue pour les deux types de codes d.o.*

J	codes doublement orthogonaux			
	sens large, taux $\frac{1}{2}$, J gen.		sens strict, taux $\frac{J}{2J}$	
	long. min. (10%)	tps à 10%	long. min. (10%)	tps à 10%
5	41 (45)	10"	98 (107)	30"
6	100 (110)	25"	323 (355)	1'
7	222 (244)	50'	879 (968)	1h30

On voit qu'obtenir un "bon" code d.o. au sens strict requiert plus de temps que pour obtenir son équivalent au sens large. Ce fait s'amplifie avec l'augmentation de J. Ainsi pour la grandeur $J = 14$, s'il nous faut 270 heures pour générer une matrice d.o. au sens strict il ne nous en faut "que" 96 pour effectuer la même chose avec un code d.o. au sens large.

5.6.3 Performances

Au chapitre des performances on représente les deux courbes 5.6 et 5.7 qui permettent de comparer l'efficacité de deux codes doublement orthogonaux "équivalents", l'un au sens large de taux $\frac{1}{2}$ comportant 8 générateurs avec des gains optimisés et l'autre au sens strict de taux $\frac{8}{16}$.

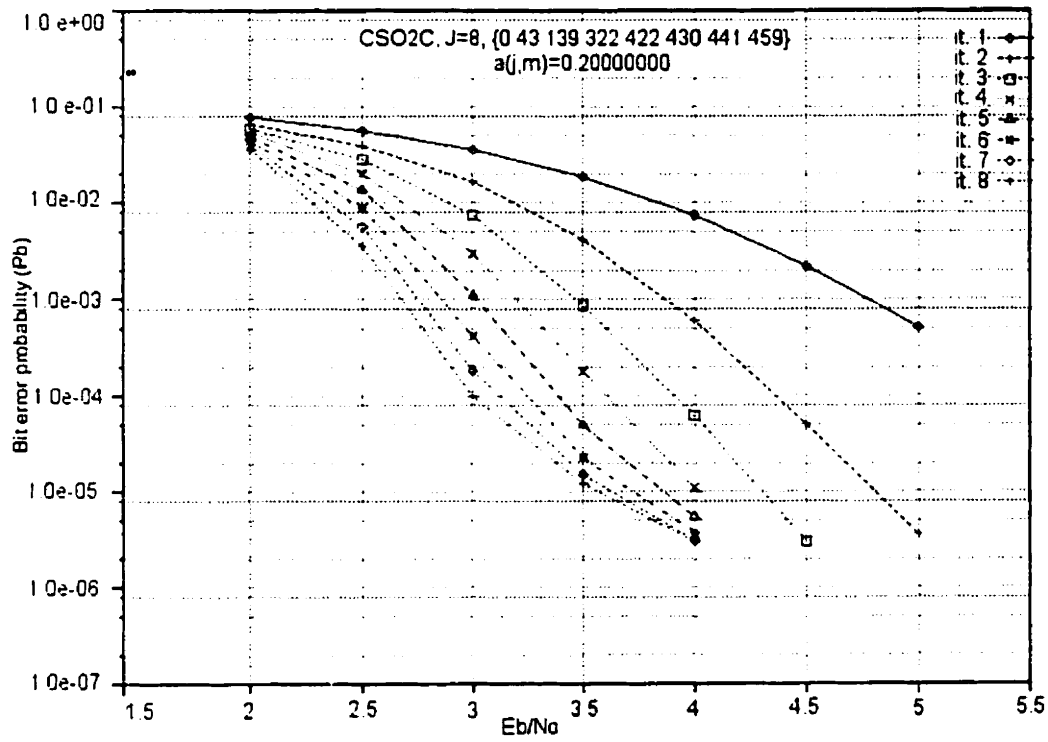

Figure 5.6 – *Probabilité d'erreur par bit en fonction de* $\frac{Eb}{No}$ *pour un code d.o. au sens large de taux* $\frac{1}{2}$ *avec* $J = 8$ *(gains "optimisés" :* $a_{j,m} = 0.2$*)*

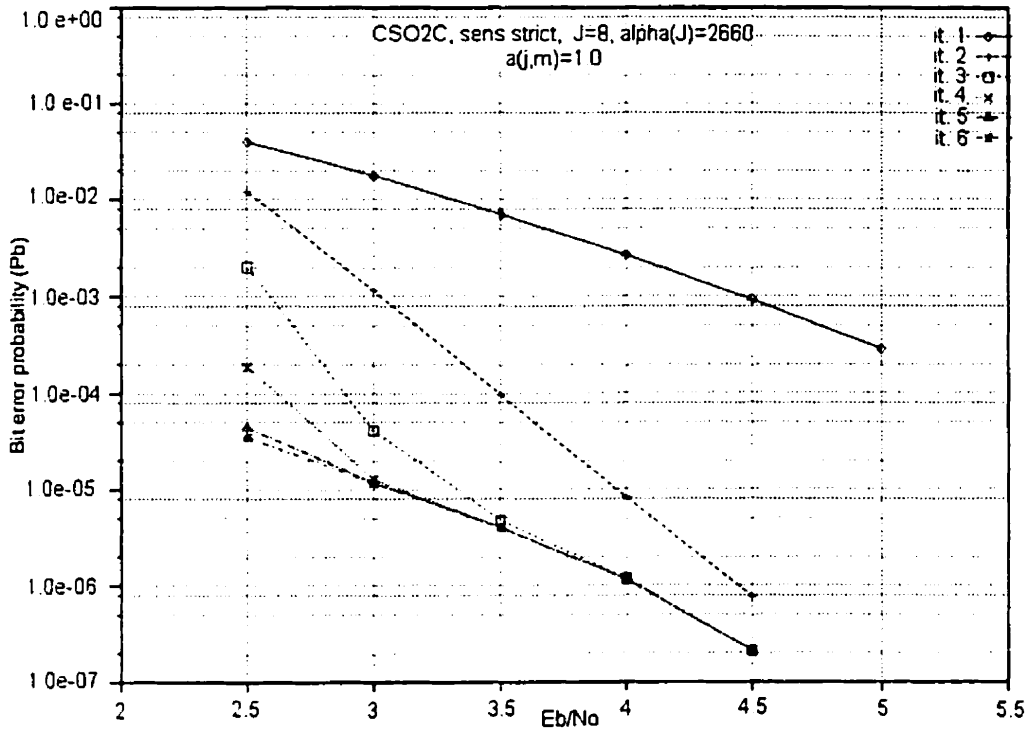

Figure 5.7 – *Probabilité d'erreur par bit en fonction de* $\frac{Eb}{No}$ *pour un code d.o. au sens strict de taux* $\frac{8}{16}$

La comparaison des deux courbes met en valeur le niveau de performances supérieur des codes d.o. au sens strict face à leurs homologues d.o. au sens large et ceci quelques soient les valeurs du rapport signal à bruit considérées. Le tableau 5.9 reprend quelques unes des valeurs les plus significatives.

Tableau 5.9 – *Valeurs significatives de comparaison d'un code d.o. au sens large de taux* $\frac{1}{2}$ *comportant 8 générateurs et d'un code d.o. au sens strict de taux* $\frac{8}{16}$

$\frac{Eb}{No}$	nb. itérations	Prob. d'erreur par bit	
		code d.o. sens large	code d.o. sens strict
2.5	4	10^{-2}	$3\ 10^{-4}$
3.5	3	10^{-5}	10^{-3}
4.5	2	10^{-4}	10^{-6}

Les codes d.o. au sens strict présentent également l'avantage d'être très peu sensibles aux réglages des gains au décodage contrairement à ceux d.o. au sens large (4.5). Ceci évite ainsi la recherche de ces coefficients et simplifie la mise en oeuvre matérielle.

De plus, les performances limites sont quasiment atteintes au bout de seulement 3 itérations réduisant ainsi de façon significative le délai au décodage [9]. Une bonne illustration de cette propriété est la suivante:

- Pour un rapport $\frac{Eb}{No}$ égal à 3.5 on obtient dans le cas du code d.o. au sens strict présenté sur les courbes précédentes une probabilité d'erreur par bit d'environ 10^{-5}. Pour aboutir à une telle valeur en utilisant un code d.o. au sens large équivalent il faudrait environ 8 itérations soit environ 3 fois plus avec des gains "optimisés" pris égaux à ceux utilisés pour générer la figure 5.6.

Il convient ici d'apporter une précision relative au comportement des codes d.o. au sens large et du réglage des gains. Il est en effet difficile de différencier par l'intermédiaire des simulations si ce sont les codes d.o. au sens large en eux même qui présentent des performances en retrait vis à vis de leurs homologues au sens strict ou si c'est la procédure de décodage utilisée qui diminue les performances d'erreur. On a vu en effet que pondérer les coefficients intervenant au sein des équations de parité dans le cas des codes d.o. au sens large permettait d'améliorer le comportement de ces codes en limitant la corrélation des symboles mis en jeu. Malheureusement, il est extrêmement compliqué de trouver les modifications optimales à adopter puisqu'il faut prendre en compte chaque facteur de pondération indépendamment à chaque itération et tenir compte du nombre total d'itérations souhaité.

5.6.4 Bilan

Fort des considérations précédentes on peut résumer les différences entre les deux types de codes en répertoriant leurs qualités et défauts respectifs :

a) Codes d.o. au sens large

. avantages

- mise en oeuvre matérielle simple
- longueur de contrainte faible

. inconvénients

- réglage des gains influant et requérant des simulations préalables
- nombre d'itérations plus élevé que dans le cas des codes d.o. au sens strict pour atteindre le même niveau de performances

b) Codes d.o. au sens strict

. avantages

- pas de réglage des gains nécessaire
- performances maximales supérieures atteintes au bout de 3 itérations

. inconvénients

- architecture matérielle complexe
- longueur des codes plus élevée que dans le cas des codes d.o. au sens large
- génération des codes plus longue

Ainsi il est fort difficile de distinguer lequel des deux types de codes sera plus ou moins bien adapté à certaines situations. Une longueur plus élevée dans le cas des codes au sens strict est certes handicapante mais elle s'avère être compensée en pratique par un nombre d'itérations moins important et de très bonnes performances notamment pour de faibles rapport signal à bruit. On pourrait toutefois préférer les codes d.o. au sens large qui, une fois les gains optimisés, possèdent des performances voisines de celles des codes d.o. au sens strict pour une longueur et une complexité matérielle moindre. Cependant l'ajustement des gains présente en lui-même de nombreuses difficultés [9].

5.7 Conclusion

Si la théorie n'est pas, une nouvelle fois, établie de manière exacte, les méthodes proposées pour générer des codes d.o. au sens strict permettent d'aboutir à des résultats satisfaisants dans des limites de temps raisonnables pour peu que J ne soit pas trop élevé. Les générateurs obtenus ici sont d'autant plus importants qu'ils ne nécessitent qu'un faible nombre d'itérations (3) pour atteindre un très bon niveau de performances. Si celui ci demeure en retrait par rapport aux codes turbo, la très faible complexité matérielle est un avantage indéniable. Le choix pratique d'utiliser des codes d.o. au sens strict ou au sens large demeure un point délicat ; chacune des deux options possèdent avantages et inconvénients.

CHAPITRE 6

CODES EN BLOCS DOUBLEMENT ORTHOGONAUX

6.1 Présentation

6.1.1 Définition

On a vu au cours du deuxième chapitre (2.2.1) une brève description du codage en blocs. Si les concepts et les techniques vues précédemment s'appliquent à des codes convolutionnels qui rappelons-le sont à la base du nouveau système de codage introduit il est possible de les élargir dans une certaine mesure aux codes en blocs.

Cette extension est réalisable en introduisant la notion de temps. Pour cela on utilise un modulo sur le paramètre j qui définissait auparavant l'instant auquel était considéré notre système (3.3.2.3). L'entier utilisé pour le modulo est naturellement la longueur du bloc d'entrée.

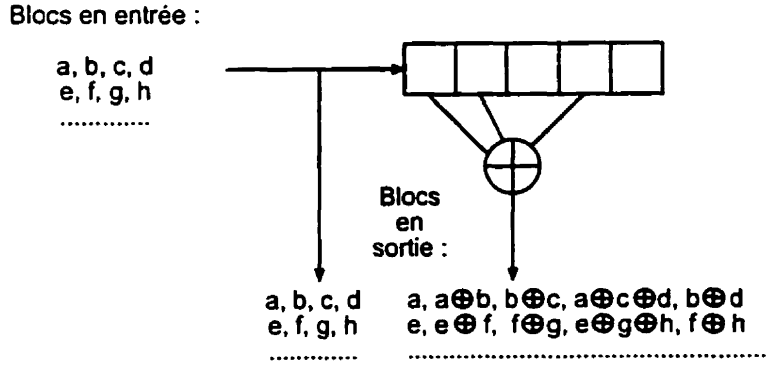

Figure 6.1 – *Exemple de structure adoptée pour l'extension aux codes en blocs*

6.1.2 Équivalence

On montre ([13]) que lorsque l'on considère ce type de code, il y a équivalence avec certaines classes de codes de faible densité de parité (en anglais: "low density parity check codes") introduits par R. G. Gallager en 1962 [8]. Si ces codes ne sont pas optimaux dans le sens de la minimisation de la probabilité d'erreur au décodage, la très simple implémentation nécessaire à leur traitement compense dans une certaine mesure cette limitation.

Pour que l'équivalence avec nos codes doublement orthogonaux considérés soit totale il faut que ces codes de faibles densités soient "deux/tiers" indépendants.

On rappelle ici quelques propriétés de ce type de codes notamment relativement à la formation de la matrice de parité.

Un code à faible densité de parité est classiquement défini par trois paramètres (u, w, x). La grandeur u représente la longueur du code, la composante w est le nombre de symboles considérés en entrée pour générer un bloc de sortie de longueur x. La matrice de parité initialement correspondante se construit selon ces paramètres avec une largeur u, un nombre de "1" par colonnes égal à w et un nombre de "1" par lignes égal à x. Cette dernière condition assure une indépendance d'un "rang" équivalente à une condition de simple orthogonalité. En modifiant quelque peu la construction de la matrice on obtient l'équivalence souhaitée au rang supérieur en procédant comme suit (soit $g_{i,j}$ les générateurs):

- on partitionne les rangées de la matrice de parité en w groupes égaux.
- seulement un "1" par colonne est autorisé dans chacun des w groupes formés.
- on considère la première colonne, on place un "1" à la $g_{0,0}ième$ ligne du premier groupe, un "1" à la $g_{0,1}ième$ ligne du second groupe, ..., un "1" à la

$g_{0,w-1}$ *ième* ligne du *wième* groupe.

- on recommence dans la colonne suivante en considérant $g_{1,0}$, $g_{1,1}$,...,$g_{1,w-1}$.
- lorsque les x premières colonnes ont été remplies l'ensemble des générateurs a été utilisé.
- les autres colonnes sont alors complétées en recopiant les x précédentes et en effectuant un décalage au niveau des rangées d'une unité au sein de chaque groupe (la dernière rangée est recopiée au sommet).

Pour fixer les idées un exemple simple est présenté ci-après :

On prend $u = 12$, $w = 2$ et $x = 3$ avec les générateurs : $g_{0,i} = \{0,0\}$, $g_{1,i} = \{0,1\}$ et $g_{2,i} = \{0,2\}$.

En suivant les étapes précédemment établies le cheminement est le suivant :

- on partage en $w = 2$ groupes les rangées.
- on remplit la première colonne du premier groupe avec un "1" à la $g_{0,0} = 0$ *ième* ligne :

1.....

.....

- sachant qu'il n'y a qu'un "1" par colonne dans chaque groupe on a (le nombre de lignes est imposé par $\frac{u}{x}$):

1.....

0.....

0.....

0.....

.....

- on poursuit avec un "1" à la $g_{0,1}$ *ième* ligne du deuxième groupe et on complète

avec des zéros :

1.....
0.....
0.....
0.....

1.....
0.....
0.....
0.....

- lorsque tous les générateurs ont été utilisés on a :

111	.....
000	.....
000	.....
000	.....
100	.....
010	.....
001	.....
000	.....

- on reproduit alors tous les groupes en décalant dans chaque groupe une ligne vers le bas jusqu'à atteindre la longueur souhaitée (ici 12)

111	000	000	000
000	111	000	000
000	000	111	000
000	000	000	111
100	000	001	010
010	100	000	001
001	010	100	000
000	001	010	100

Il est possible d'établir une formule générique de positionnement des "1" par colonne en utilisant la formule suivante :

$$l_s = \frac{s * u}{x} + [(\frac{i_c - i_r}{x} + g_{i_r,s}) mod(-\frac{u}{x})] \tag{6.1}$$

où i_c est le numéro de la colonne qui nous préoccupe et $i_r = i_c mod(u)$ avec s variant de 0 à $w - 1$.

Exemple illustratif : reprenons notre code précédemment défini et étudions comme cas particulier la colonne 7. On a ainsi :
$l_0 = \frac{0*12}{3} + [(\frac{7-7mod(3)}{3} + g_{7mod(3),0}) mod(\frac{-12}{3})]$ et $l_1 = \frac{12}{3} + [(\frac{7-7mod(3)}{3} + g_{7mod(3),1}) mod(\frac{-12}{3})]$
Soit :
$l_0 = 2$ et $l_1 = 7$
Ceci est conforme avec la matrice précédemment obtenue.

Remarque : les générateurs utilisés ici ne servaient qu'à illustrer la construction de la matrice de parité et ne constituent pas un code doublement orthogonal.

La condition de double orthogonalité qui se détermine dans ce cas à partir de celle de deux/tiers indépendance des codes de faible densité de parité est la suivante

$$\begin{array}{c} \textit{soient } i_r \textit{ et } i_r'', \forall\ s, s' \textit{ et } i_r' \textit{ tels que } (s \neq s', i_r \neq i_r' \textit{ et } i_r' \neq i_r'') \\ \textit{les combinaisons } (g_{i_r,s} - g_{i_r',s}) - (g_{i_r'',s'} - g_{i_r',s''}) \textit{ sont distinctes.} \end{array} \tag{6.2}$$

La forme de l'équation permet de faire immédiatement le parallèle avec ce qui a déjà été vu dans le cas des codes au sens strict. La condition garantissant

l'équivalence était alors :

$$\begin{gathered} Soit(k,v),\ \forall\ (l,m,n)\ tels\ que\ n \neq l,\ m \neq k\ et\ m \neq v\ on\ a : \\ g_{k,l} - g_{m,l} + g_{m,n} - g_{v,n}\ distincts. \end{gathered} \tag{6.3}$$

On retrouve exactement la même forme que précédemment en remplaçant respectivement k et v par i_r et $i_{r'}$ et l, m, n par s, i'_r, s''.

Cette remarque est on s'en doute très importante puisqu'elle va grandement faciliter la génération de ce type de codes. Il suffira en effet d'examiner les possibilités d'extension qu'offrent les méthodes étudiées lors de l'analyse des codes au sens strict.

6.2 Génération

La seule restriction qui empêche l'application directe des techniques mises en place pour les codes au sens strict concerne les dimensions. En effet nous ne sommes plus ici en présence d'une matrice de générateurs "carrée" et nous sommes en droit de nous demander si cela peut influencer de quelque manière que ce soit l'application des méthodes précédemment usitées. Une brève analyse prouve que non et avec quelques modifications simples, les méthodes demeurent applicables.

On utilisera donc comme principe de génération une méthode pseudo-aléatoire telle que décrite dans le chapitre précédent. Les mêmes constations proposées pour les codes au sens strict s'appliquent ici.

6.3 Réduction

La technique de réduction est exactement similaire à celle exposée dans le cas des codes au sens strict. La situation d'arrêt demeure la même avec la présence d'un cycle et l'algorithme pour y aboutir demeure inchangé.

6.4 Valeurs générées

Les codes de taux $\frac{x}{u+x}$ ont été générés pour des valeurs de x et u variant de 2 à 10.

Il peut être judicieux de remarquer que la transposée de la matrice de générateurs doublement orthogonaux d'un code de taux $\frac{x}{x+u}$ donne un ensemble de générateurs doublement orthogonaux de taux $\frac{u}{u+x}$.

Il est possible d'établir relativement simplement la preuve de cela:

Soit $g_{i,j}$ des générateurs vérifiant la condition de double orthogonalité. On note $h_{i,j} = g_{j,i}$.

On veut montrer que:

soit i_r, i''_r, $\forall$ $(s,s',i_{r'})$, tels que $s \neq s'$, $i_{r'} \neq i_r$, $i_{r'} \neq i_{r''}$ et $\forall$ $(v,v',i_{r'''})$, tels que $v \neq v'$, $i_{r'''} \neq i_r$, $i_{r'''} \neq i_{r''}$ et $(i'_r,s,s') \neq (i'''_r,v,v')$ on a:

$$h_{i_r,s} - h_{i'_r,s} + h_{i'_r,s'} - h_{i'''_r,s'} \neq h_{i_r,v} - h_{i'''_r,v} + h_{i'''_r,v'} - h_{i''_r,v'}. \tag{6.4}$$

Procédons par équivalence en modifiant cette inéquation dont on ne sait pas encore

si elle est vérifiée.

$$\Leftrightarrow \; g_{s,i_r} - g_{s,i'_r} + g_{s',i'_r} - g_{s',i''_r} \neq g_{v,i_r} - g_{v,i'_r} + g_{v',i'_r} - g_{v',i''_r} \tag{6.5}$$

$$\Leftrightarrow \; g_{s,i_r} - g_{v,i_r} + g_{v,i'''_r} - g_{v',i'''_r} \neq g_{s,i'_r} - g_{s',i'_r} + g_{s',i''_r} - g_{v',i''_r} \tag{6.6}$$

avec $i'''_r \neq i_r$, $i'_r \neq i''_r$, $s \neq s'$, $v \neq v'$.

- Si $s' \neq v'$ et $s \neq v$ alors on retrouve :

$$\Leftrightarrow \; g_{s,i_r} - g_{v,i_r} + g_{v,i'''_r} - g_{v',i'''_r} \neq g_{s,i'_r} - g_{s',i'_r} + g_{s',i''_r} - g_{v',i''_r} \tag{6.7}$$

avec $i'''_r \neq i_r$, $i'_r \neq i''_r$, $s \neq s'$, $v \neq v'$, $s' \neq v'$ et $s \neq v$.

Ce qui est l'expression de la condition de double orthogonalité que l'on sait être vérifiée par $g_{i,j}$. La proposition est alors bien établie.

- Si $s' = v'$ on arrive à l'équivalence :

$$\Leftrightarrow \; g_{s,i_r} - g_{v,i_r} + g_{v,i'''_r} - g_{v',i'''_r} \neq g_{s,i'_r} - g_{s',i'_r} \tag{6.8}$$

$$\Leftrightarrow \; g_{s,i_r} - g_{v,i_r} + g_{v,i'''_r} \neq g_{s,i'_r} - g_{s',i'_r} + g_{s',i'''_r} \tag{6.9}$$

$$\Leftrightarrow \; g_{s,i_r} - g_{v,i_r} + g_{v,i'''_r} - g_{s,i'''_r} \neq g_{s,i'_r} - g_{s',i'_r} + g_{s',i'''_r} - g_{s,i'''_r} \tag{6.10}$$

avec $s \neq s'$ et $i_r \neq i'''_r$

+ si $s = v$

$$\Leftrightarrow \; g_{s',i'_r} - g_{s,i'_r} \neq g_{s',i'''_r} - g_{s,i'''_r} \tag{6.11}$$

On a forcément $i'_r \neq i'''_r$ sinon $(i'_r,s,s') = (i'''_r,v,v')$

$$\Leftrightarrow \; g_{s',i'_r} - g_{s,i'_r} + g_{s,i''_r} - g_{s,i''_r} \neq g_{s',i'''_r} - g_{s,i'''_r} + g_{s,i''_r} - g_{s,i''_r} \tag{6.12}$$

avec $s \neq s'$ et $i'_r \neq i''_r$ et $i''_r \neq i'''_r$

Cette inégalité est forcément vérifiée par les $g_{i,j}$ (elle est incluse dans celles imposées par la double orthogonalité) et donc

+ si $s \neq v$

on a alors

$$\Leftrightarrow \; g_{s,i_r} - g_{v,i_r} + g_{v,i''_r} - g_{s,i''_r} \neq g_{s,i'_r} - g_{s',i'_r} + g_{s',i''_r} - g_{s,i''_r} \tag{6.13}$$

avec $s \neq s'$, $i_r \neq i'''_r$ et $s \neq v$.

Pour les mêmes raisons que précédemment cette inéquation est vérifiée par les $g_{i,j}$.

- Si $s = v$ on effectue un raisonnement identique.

Ainsi $\{g_{i,j}\}$ vérifient la condition de double orthogonalité

$\Leftrightarrow \{h_{i,j}\} = \{g_{j,i}\}$ vérifient la condition de double orthogonalité.

Le tableau ci dessous présente pour différents taux de codage du type $\frac{x}{x+u}$ les longueurs minimales obtenues.

Tableau 6.1 – *Longueurs minimales obtenues pour des codes en blocs de taux* $\frac{x}{x+u}$

		x								
		2	3	4	5	6	7	8	9	10
u	2	1	2	3	6	9	15	19	28	37
	3	2	5	11	21	36	57	88	124	178
	4	3	11	25	54	86	149	239	343	467
	5	6	21	54	104	221	353	516	784	1135
	6	9	36	86	221	363	664	1053	1561	2090
	7	15	57	149	353	664	1102	1856	2634	3612
	8	19	88	239	516	1053	1856	2660	4333	5874
	9	28	124	343	784	1561	2634	4333	5707	9382
	10	37	178	467	1135	2090	3612	5874	9382	12426

On retrouve naturellement sur la diagonale de cette matrice des valeurs connues correspondant aux codes au sens strict de taux $\frac{w}{w+w} = \frac{w}{2w}$. On remarque également la forme symétrique de la matrice suite à la remarque sur la transposition exposée précédemment.

Les tableaux contenant tous les générateurs pour les différents taux étudiés se trouvent à l'Annexe IV.

CHAPITRE 7

CONCLUSION ET OUVERTURE SUR TRAVAUX FUTURS

7.1 Bilan de l'étude réalisée

Ce travail de recherche s'est inscrit dans la prolongation des travaux précédemment effectués ayant abouti à la mise au point d'un nouveau système de codage et de décodage.

Notre étude a permis de mettre en évidence et de démontrer certaines propriétés mathématiques des codes doublement orthogonaux utilisés par ce système. Celles-ci nous ont donné la possibilité de mieux appréhender ces ensembles et de différencier deux types de codes: ceux doublement orthogonaux au sens large et ceux doublement orthogonaux au sens strict. Nous nous sommes ensuite concentrés à mettre en oeuvre pour chacun des deux cas des méthodes de génération et de réduction efficaces dans le but d'aboutir au système le plus performant possible sur le plan de la latence et du délai. Les résultats générés qui définissent de nouveaux ensembles doublement orthogonaux utilisables en pratique font apparaître de très importantes améliorations par rapport aux valeurs précédemment connues. Certaines grandeurs n'avaient d'ailleurs pu être établies faute de techniques viables. On peut donc considérer que le but de ce travail de recherche a été atteint. Il nous faut cependant rester conscient du fait que si les techniques employées sont simples et performantes, il ne nous a pas été possible d'aboutir aux ensembles minimaux théoriques pour un grand nombre d'éléments.

Pratiquement, des codes doublement orthogonaux au sens large de taux $\frac{1}{2}$ de longueur réduite ont été générés pour un nombre d'éléments J inférieur ou égal à

10 dans le cas des ensembles sans les négatifs, inférieur ou égal à 15 dans le cas des ensembles complets. Des codes doublement orthogonaux au sens strict de taux $\frac{J}{2J}$ de longueur réduite ont été établis pour J variant de 2 à 14. Enfin nous nous sommes attachés à obtenir des codes en blocs doublement orthogonaux de longueur réduite de taux $\frac{I}{I+J}$ pour I et J variant de 2 à 10.

Différents aspects relatifs à nos techniques d'obtention de ces codes ont été étudiés avec notamment une étude de la longueur minimale et une analyse temporelle.

Des résultats de simulation ont été interprétés afin de mieux cerner les performances de ces codes et de bien distinguer leurs différents types.

7.2 Améliorations envisageables

Si le sujet consistant à trouver des ensembles doublement orthogonaux paraît simple de prime abord, les difficultés théoriques et l'importance des calculs mis en jeu confèrent à ce problème une complexité inattendue. Ce fait qui n'en diminue pas l'intérêt de son étude (bien au contraire) assure d'avoir encore de nombreuses pistes à explorer. On peut ainsi poser les bases d'une étude future en proposant plusieurs domaines dans lesquels il serait intéressant d'approfondir le travail présenté dans ce mémoire.

- Aspect théorique

  Ce premier point est le plus naturel puisque déterminer une théorie mathématique permettant de trouver ces ensembles doublement orthogonaux compacts résoudrait le problème. Celle-ci, si elle existe, ne sera sans aucun doute non tri-

viale. Rien que déterminer la longueur minimale "atteignable" serait un atout indéniable pour aider à la réduction de ces ensembles. Nos tentatives dans ce domaine n'ont permis que d'obtenir des grandeurs par valeurs inférieures ou supérieures.

- Aspect algorithmique

Ce point s'inscrit dans la complémentarité de celui évoqué précédemment. L'absence de théorie exacte oblige à élaborer différentes stratégies pour espérer se rapprocher le plus possible dans un temps minimum de l'ensemble doublement orthogonal minimal. Notre étude à explorer principalement le champ de techniques basées sur l'utilisation d'un paramètre aléatoire. D'une façon générale ce type de choix produit des résultats intéressants même si on ne peut jamais réellement savoir si l'ensemble obtenu se trouve "éloigné" ou non de celui optimal. A défaut d'obtenir une théorie de génération exacte, il serait très intéressant d'établir une stratégie permettant d'aboutir de manière certaine au résultat cherché sans pour autant imposer un nombre de calcul astronomique.

- Aspect matériel

L'évolution du matériel informatique notamment celle des processeurs est telle que dans quelques mois il sera possible d'utiliser les méthodes exposées dans ce mémoire pour se rendre à des ensembles avec un nombre d'éléments plus important sans demander un temps de calcul excessif.

7.3 Ouverture

La génération de ces différents codes doublement orthogonaux ont permis d'apporter des améliorations significatives au niveau de la latence et de la mémoire nécessaire au bon fonctionnement du nouveau système de codage. Ce dernier apparaît ainsi comme étant une alternative tout à fait crédible aux systèmes actuellement utilisés. On pense notamment aux codeurs et décodeurs turbo qui, s'ils disposent de performances supérieures, présentent une plus grande complexité et un plus long délai de traitement.

RÉFÉRENCES

[1] MASSEY, J.L., "Threshold Decoding", *MIT Press*, Cambridge, MA., 1963.

[2] WOZENCRAFT, J.M. et REIFFEN, B., "Sequential Decoding", *Res. Monograph 10*, Cambridge, Mass.: MIT press, 1961.

[3] FORNEY, G.D., Jr, "Concatenated Codes", *MIT Press*, Cambridge, Mass., 1966.

[4] GAAL, L., "Classical Galois Theory with Examples", *Chelsea publishing company*, 3rd edition, New York, N.Y., 1979.

[5] GAGNON, F., HACCOUN, D., BATANI, N. et CARDINAL, C., "Apparatus for Convolutional Self-Doubly Orthogonal Encoding and Decoding", *US Patent Application Filed*, 2 octobre 1997, US Patent and Trademark Office, Washington DC, and *European Patent Application Filed*, 2 octobre 1998, European Patent Office, Netherlands.

[6] BENEDETTO, S. et MONTORSI, G., "Unveiling Turbo Codes: Some Results on Parallel Concatenated Coding Schemes", *IEEE Transactions on Information Theory*, vol. 43, pp. 409-428, mars 1996.

[7] ROYER, G., "Évaluation des entrelaceurs au sein des Codes Turbo par simulations", *Mémoire de Maîtrise*, École Polytechnique de Montréal, juin 2000.

[8] GALLAGER, R.G., "Low-Density Parity-Check Codes", *IRE Transactions on Information Theory*, janvier 1962.

[9] CARDINAL, C., HACCOUN, D., GAGNON, F et BATANI, N., "Turbo Decoding Using Convolutional Self Doubly Orthogonal Codes", *ICC*, pp. 113-117, Vancouver, B.C., 1999.

[10] BERROU, C. et GLAVIEUX, A., "Refecting on the Prize Paper - Near-Optimum Error-Correcting Coding and Decoding: Turbo-Codes", *IEEE Information Theory Society Letter*, vol. 48, juin 1998.

[11] WU, W.W., "New Convolutionnal Codes, Part I", *IEEE Transactions on Communications*, vol. COM-23, pp. 942-956, septembre 1975.

[12] CARDINAL, C., HACCOUN, D., GAGNON, F. et BATANI, N., "Convolutional Self-Doubly Orthogonal Codes for Iterative Decoding without Interleaving", *IEEE International Symposium on Information Theory, 1998*, Cambridge, MA., août 1998.

[13] GAGNON, F. et HACCOUN, D., "Convolutional Codes with Extended Self-Orthogonality and new Low-Density Parity-Check Block Codes", soumis pour publication, *IEEE Trans. Inform. Theory*, 1999.

[14] SHANNON, C.E., "A Mathematical Theory of Communication", *Bell System Technical Journal*, vol. 27, pp. 379-423, 623-656, 1948.

[15] HAMMING, R.W., "Error Detecting and Error Correcting Codes", *Bell System Technical Journal*, avril 1950.

[16] GOLAY, M.J.E., "Notes on Digital Coding", *Proceedings of the IRE*, juin 1949.

[17] ELIAS. P., "Coding for Noisy Channels", *IRE conv. Rec.*, pt.4, pp. 37-46, 1955.

[18] FANO. R.M., "A Heuristic Discussion of Probabilistic Decoding", *IEEE Trans. Inform. Theory*, vol. IT-9, pp. 64-75, avril 63; *Math. Rev.*, vol. 29, p. 3283.

[19] VITERBI, A.J., "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm", *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 260-269, avril 1967.

[20] FORNEY, G.D., Jr., "Convolutional Codes I: Algebraic structure", *IEEE Trans. Inform. Theory*, vol. IT-16, no. 6, pp. 720-738, novembre 1970.

[21] FORNEY, G.D., jr., "Convolutional Codes II: Maximum Likelihood Decoding", and "Convolutional Codes III: Sequential Decoding", *Inform. Syst. Lab.*, Standford University, Standford, Calif., Tech. Rep. 7004-1, 1972.

[22] BERROU, C., GLAVIEUX, A. et THITIMAJSHIMA, P., "Near Shannon Limit Error Correcting Coding and Decoding: Turbo Codes", *International Conference on Communications (ICC'93)*, pp. 1064-1070, Genève, Suisse, 1993.

[23] STIFFLER, J.J., "Synchronization in Communication Systems", *Englewood Cliffs*, New Jersey: Prentice-Hall, 1969.

[24] CHAN, F. et HACCOUN, D., "Adaptive Decoding of Convolutional Codes", *Canadian Conf. on Electrical and Computer Engineering*, 69.4.1-69.4.4, Québec, Canada, 1991.

[25] BELZILE, J., "Décodage sous-optimal des codes convolutionnels et applications", *thèse de PhD*, Ecole Polytechnique de Montréal, 1993.

[26] BAHL, L., COCKE, J., JELINEK, F. et RAVIV, J., "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate", *IEEE Int. Symp. Inform. Theory*, p. 90, Asimolar, Ca, 1972.

[27] BERLEKAMP, E.R., "Key Papers in the Development of Coding Theory", *IEEE Press*, pp. 67, 1974.

[28] BOSE, R.C. et RAY-CHAUDHURI, D.K., "On a Class of Error Correcting Binary Group Codes", *Inform. Control*, vol. 3, pp. 68-79, mars 1960.

[29] BOSE, R.C. et RAY-CHAUDHURI, D.K., "Further Results in Error Correcting Binary Group Codes", *Inform Control*, vol. 3, pp. 279-290, septembre 1960.

[30] HOCQUENGHEM, A., "Codes correcteurs d'erreurs", *Chiffres*, vol. 2, pp. 147-156, 1959.

[31] UNGERBOECK, G., "Channel Coding with Multi-level/phase Signals", *IEEE Trans. Inf. Theory*, pp. 55-67, janvier 1982.

[32] HAGENAUER, J. et HOEHER, P., "A Viterbi Algorithm with Soft-Decision Outputs and its Applications", *GLOBECOM 89*, pp. 1680-1686, Dallas, Texas, 1989.

[33] HAGENAUER, P., ROBERTSON, P. et PAPKE, L., "Iterative (turbo) Decoding of Systematic Convolutional Codes with the MAP and the SOVA Algorithms", *ITG Tagung*, pp. 21-29, Frankfort, Allemagne, 1994.

[34] HAGENAUER,J., OFFER, E. et PAPKE, L., "Iterative Decoding of Binary Bloc and Convolutional Codes", *IEEE Trans. Inform Theory*, vol. 42-2, pp. 429-445, 1996.

[35] CHANG R.W. et HANCOCK, J.C., "On Receiver Structure for Channels having Memory", *IEEE Trans. Inform. Theory*, vol. IT-12, pp. 463-468, 1966.

[36] ELIAS, P., "Error Free Coding", *IRE Trans. on Inform. Theory*, vol. PGIT-4, pp. 29-37, 1954.

[37] IMAI, H. et HIRAKAWA, S., "A new Multilevel Coding Method using Error-Correcting Codes", *IEEE Trans on Inform. Theory*, vol. IT-23, pp. 371-377, 1997.

[38] PYNDIAH, R., GLAVIEUX, A., PICART, A. et JACQ, S., "Near Optimum Decoding of Products Codes", *IEEE GLOBECOM '94 Conference*, vol. 1/3, pp. 339-343, San Francisco, nov. - dec. 1994.

[39] PEREZ, L., SEGHERS, J. et COSTELLO, D.J, "A Distance Spectrum Interpretation of Turbo Codes", *IEEE Trans. Inform. Theory*, vol. 42-6, pp. 1698-1709, 1996.

[40] ROBERTSON, P., "Improving Decoder and Code Structure of Parallel Concatenated Recursive Systematic (Turbo) Codes", *Int. Conf. Universal Personal Commun., pp. 183-187, San Diego, CA, 1994.*

[41] BENEDETTO, S., MONTORSI, G., DIVSALAR, D. et POLLARA, F., "Soft-Ouput Decoding Algorithms in Iterative Decoding of Turbo Codes", *JPL TDA Progress Report, vol. 42-124, pp. 63-87, 1996.*

[42] PIETROBON, S.S. et BARBULESCU, S.A., "A Simplification of the Modified Bahl Decoding Algorithm for Systematic Convolutional Codes", *Int. Symp. Inform. Theory & its applications*, pp. 1073-1077, Sydney, Australie, 1994.

[43] ROBERTSON, P., VILLEBRUN, E. et HOEHER, P., "A Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain", *ICC 95*, pp. 1009-1013, Seattle, WA, 1995.

[44] BAECHLER, B., HACCOUN, D. et GAGNON, F., "On the Search for Convolutional Self-Doubly Orthogonal Codes", accepté pour présentation, *IEEE Int. Symp. Inform. Theory*, Sorrento, Italie, juin 2000.

[45] ROBINSON, J.P. et BERNSTEIN, A.J., "A Class of Binary Recurrent Codes with Limited Error Propagation", *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 106-113, janvier 1967.

[46] WU, W.W., "New Convolutional Codes - Part I", *IEEE Trans. Inform. Theory*, vol. COM-23, no. 9, septembre 1975.

[47] RUDOLPH, L.D., "A Class of Majority Logic Decodable Codes", *IEEE Trans. Inform. Theory*, vol. IT -13, pp. 305-307, avril 1967.

[48] CUONG HOA, S., "Construction de règles de Golomb optimales", *Mémoire de Maîtrise*, École Polytechnique de Montréal, 1999.

[49] BOUZOUITA, N., "Sur le décodage itératif des codes turbo", *Mémoire de Maîtrise*, École Polytechnique de Montréal, juin 1997.

[50] Codage turbo
http://www-sc.enst-bretagne.fr/turbo/principale.html

[51] Générateurs aléatoires
http://www.npac.syr.edu/projects/random/brief.html

[52] Séquences de Sidon
http://www.mathsoft.com/asolve/constant/erdos/erdos.html

[53] Règles de Golomb
http://www.research.ibm.com/people/s/shearer/grule.html

Annexe I

Simplification de la condition de double orthogonalité

On va montrer que pour un ensembles de plus de quatre éléments, la condition 3.7 implique les deux autres (3.5 et 3.6). On considérera $\{0, g_1, ..., g_{J-1}\}$ un ensemble doublement orthogonal de J éléments.

Montrons tout d'abord que 3.7 $\rightarrow$ 3.5 en raisonnant par l'absurde.

Soit (k,l,k',l') tel que $k > l$, $k' > l'$, $(k, l) \neq (k', l')$ et $g_k - g_l = g_{k'} - g_{l'}$.
Ajoutons de part et d'autre g_m tel que $g_m = max(g_k, g_{k'})$. On obtient : $g_k - g_l + g_m = g_{k'} - g_{l'} + g_m$.
Retranchons alors $g_n = g_0 = 0$ pour aboutir à :

$$g_m - g_l + g_k - g_n = g_m - g_{l'} + g_{k'} - g_n$$

$$avec \begin{cases} m > l \\ m \geq k \\ k > n \\ l \neq k \\ l \geq n \end{cases} \quad \begin{cases} m' > l' \\ m' \geq k' \\ k' > n' \\ l' \neq k' \\ l' \geq n' \end{cases} \qquad (l, k) \neq (l', k') \qquad \text{(I.1)}$$

On peut détailler les inégalités concernant les indices :

. $g_m = max(g_k, g_{k'}) \rightarrow \begin{array}{l} g_m \geq g_k > g_l \rightarrow m > l \\ g_m \geq g_{k'} > g_{l'} \rightarrow m' > l' \end{array}$

. $k > l \rightarrow k \neq l$

. $k' > l' \rightarrow k' \neq l'$

. $g_m \geq g_k \rightarrow m \geq k$

. $g_m \geq g_{k'} \rightarrow m \geq k'$

$k > l \geq g_n \rightarrow l \geq n,\ k > n$

$k' > l' \geq g_n \rightarrow l' \geq n',\ k' > n'$

. $(m, l, k, n) \neq (m', l', k', n')$ car $(l, k) \neq (l', k')$

Ainsi si l'on suppose qu'il existe (k,l,k',l') avec $k > l$, $k' > l'$ et $(l, k) \neq (l', k')$ tel que: $g_k - g_l = g_{k'} - g_{l'}$ alors il existe (m,l,k,n) et (m',l',k',n') tels que $m > l$, $m \geq k$, $k > n$, $l \neq k$, $l \geq n$; $m' > l'$, $m' \geq k'$, $k' > n'$, $l' \neq k'$, $l' \geq n'$ et $(m, l, k, n) \neq (m', l', k', n')$ tel que: $g_m - g_l + g_k - g_n = g_{m'} - g_{l'} + g_{k'} - g_{n'}$
Ce qui est contraire à 3.7 et donc 3.7 → 3.5.

Montrons maintenant que 3.7 implique 3.6.
On suppose qu'il existe (k,l,m,n,k',l') avec $k > l$, $l \geq n$, $m > n$, $l \neq m$, $k \geq m$, $k' > l'$ tel que: $g_k - g_l + g_m - g_n = g_{k'} - g_{l'}$.

. <u>On suppose $g_n \neq g_0 = 0$</u>

- On a alors forcément $g_n \neq g_{l'}$ sinon si $g_n = g_{l'}$ alors $g_k - g_l + g_m = g_{k'}$ soit $g_k - g_l = g_{k'} - g_m$ avec $k > l$, $k' > m$ et $k \neq k'$ car $l \neq m$. Situation impossible d'aprés la condition 3.5.
- Ajoutons g_n. On aboutit à: $g_k - g_l + g_m = g_{k'} - g_{l'} + g_n$
- Retranchons g_0: $g_k - g_l + g_m - g_0 = g_{k'} - g_{l'} + g_n - g_0$
On a ainsi: $k > l$, $l \neq m$, $g_m > g_n > g_0$, $g_l \geq g_0$, $k \geq m$.
* si $k' \geq n$, on a $k' \geq n$, $g_n > g_0$, $g_{l'} = g_n$, $k' > l'$, $g_{l'} \geq g_0$
* si $n > k'$, $n \geq k'$, $n \geq k' > l' \rightarrow a_n > a_{l'}$, $g_{l'} \geq g_0$, $g_{k'} > g_{l'} \rightarrow k' \neq l'$, $g_{k'} > g_{l'} \geq g_0$ $\rightarrow g_{k'} > g_0$
Les deux situations sont impossibles d'après 3.7.

. <u>On suppose $g_n = g_0 = 0$</u>

On a alors, $g_k - g_l + g_{+m} - g_0 = g_{k'} - g_{l'}$.
- On a nécessairement $g_l \neq g_{l'}$ sinon $g_k - g_0 = g_{k'} - g_m$ avec $g_k > g_0$, $g_{k'} > g_m$, $g_k \neq g_{k'}$ car $g_m \neq g_0$. Ceci est impossible d'après 3.7.

- Ajoutons g_l : $g_k + g_n - g_0 = g_{k'} - g_{l'} + g_l$

* si $g_l \neq g_0$, on retranche g_0 pour obtenir : $g_k - g_0 + g_m - g_0 = g_{k'} - g_{l'} + g_l - g_0$ avec $g_k > g_0$, $g_k \geq (g_m)$, $g_m > g_0$, $g_m \neq g_0$.

** si $g_{k'} \geq g_l$, $g_k \geq g_l$, $g_{k'} > g_{l'}$, $g_{l'} \neq g_l$ (cf avant) $g_l > g_0$ et $g_{l'} \geq g_0$

** si $g_l > g_{k'}$ $(g_l - g_{l'} + g_{k'} - g_0)$, $g_l > g_{k'}$, $g_l > g_k > g_{l'} \rightarrow g_l > g_{l'}$ $g_{k'} > g_{l'} \rightarrow g_{l'} \neq g_{k'}$, $g_{k'} > g_0$ et $a_{l'} > a_0$.

Les deux situations sont impossibles d'après 3.7.

* si $g_l = g_0$, on obtient alors une expression du type : $g_k - g_0 + g_m - g_0 = g_{k'} - g_{l'}$ (avec $g_0 = 0$ (imposé par $g_l = 0$)), on a : $g_k + g_m = g_{k'} - g_{l'}$ avec $k' > l'$, $k \geq m$, $\rightarrow$ $g_{k'} = g_k + g_m + g_l \rightarrow k' > (k, m, l')$.

On dénombre alors cinq cas :

- $g_{l'} > max(g_k, g_m) = g_k$

on a alors : $g_k + 0 + g_k + 0 = 2.g_k$ et $g_{k'} - g_{l'} + g_k - g_m = 2.g_k$.

Les deux expressions sont donc égales avec des relations entre les indices : $k' > l'$, $k' \geq l'$, $k > g_m$, $k \neq l'$ et $g_{l'} \geq g_m (g_{l'} > max(g_k, g_m)$ ce qui fait apparaître une contradiction avec la condition 3.7.

- $g_{l'} < min(g_k, g_m) = g_m$

on a alors : $g_{k'} + 0 + g_k + 0 = g_k + g_{k'}$ et $g_{k'} - g_m + g_{k'} - g_{l'} = g_{k'} - g_m + g_k +$.

Les deux expressions sont donc égales avec des relations entre les indices : $k' > l'$, $k' \geq l'$, $k > g_m$, $k \neq l'$ et $g_{l'} \geq g_m (g_{l'} > max(g_k, g_m)$ ce qui fait apparaître une contradiction avec la condition 3.7.

- $g_m < g_{l'} < g_k$

on obtient : $g_k - 0 + g_k - 0 = 2.g_k$ et $g_{k'} - g_{l'} + g_k - g_m = g_k + g_m + g_k - g_m = 2.g_k$.

Les deux égalités précédentes sont donc équivalentes avec $k' \geq k$, $k' > l'$, $g_{l'} \geq g_m$, $g_l \neq g_k$ et $g_k > g_m$ ce qui est non conforme à la condition 3.7.

- $g_m = g_{l'}$

on retrouve également l'égalité entre : $g_k - 0 + g_k - 0 = 2.g_k$ et $g_{k'} - g_{l'} + g_k - g_m = 2.g_k$ avec $k' > l'$, $k \geq k'$, $g_k > g_m$, $g_{l'} = g_m \neq g_k$ et $g_{l'} \geq g_m$. Une nouvelle fois, ceci est

contraire à 3.7.

- $g_k = g_{l'}$ on arrive à : $g_{k'} - 0 + g_k - 0 = g_k + g_{k'}$ et $g_{k'} - g_{l'} + g_{k'} - g_l = g_{k'} - g_l + g_k + g_l = g_k + g_{k'}$ avec $k' > l'$, $k' \geq k$, $g_{k'} > g_l$, $g_{l'} = g_k > g_l$ et $g_{l'} = g_k \neq g_{k'}$ ce qui est une nouvelle fois en contradiction avec 3.7.

Ainsi dans tous les cas on aboutit à une contradiction. On a donc réussi à prouver l'implication : 3.7 $\rightarrow$ 3.6. En conclusion, pour des ensembles d'au moins quatre éléments, la condition de double orthogonalité se résume à l'obtention de :

$$\left\{\begin{array}{l} k > l \\ k \geq m \\ m > n \\ l \geq n \\ l \neq m \end{array}\right. \quad \left\{\begin{array}{l} k' > l' \\ k' \geq m' \\ m' > n' \\ l' \geq n' \\ l \neq m \end{array}\right. \quad (k, l, m, n) \neq (k', l', m', n') \qquad \text{(I.2)}$$

$$g_k - g_l + g_m - g_n \neq g_{k'} - g_{l'} + g_{m'} - g_{n'}$$

Annexe II

Équivalence différences - sommes

Notre but est d'établir l'équivalence entre les deux propositions :

Soit $\{0, g_1, ..., g_{J-1}\}$ un ensemble d'entiers distincts

$$\forall(i, j, k, l, i', j', k', l') \text{ tels que } \begin{cases} i \geq k \\ j \geq l \\ i \neq k \\ k \neq l \\ l \neq i \end{cases} \begin{cases} i' \geq k' \\ j' \geq l' \\ i' \neq k' \\ k' \neq l' \\ l' \neq i' \end{cases} (i, j, k, l) \neq (i', j', k', l')$$

$$g_i - g_j + g_k - g_l \neq g_{i'} - g_{j'} + g_{k'} - g_{l'} \tag{II.1}$$

$$\begin{gathered} \forall(p, q, r, s, p', q', r', s') \text{ tel que} \\ (p, q, r, s) \text{ ne soit pas une permutation de } (p', q', r', s') \\ g_p + g_q + g_r + g_s \neq g_{p'} + g_{q'} + g_{r'} + g_{s'} \end{gathered} \tag{II.2}$$

Montrons tout d'abord que $II.2 \rightarrow II.1$

L'implication est évidente dès lors que l'on parvient à prouver que les restrictions de permutations de la proposition 2 sont incluses dans celles de 1. Pour établir cela, il est possible d'étudier tous les cas en se demandant s'il existe (i, j, k, l, i', j', k', l') tels que les conditions de 1 soient vérifiées et que l'on ait (i, j, k, l) permutation de (i', j', k', l').

Si :

(i,j',k,l')=(i',j,k',l) → (i,j,k,l)=(i',j',k',l') Impossible

(i,j',k,l')=(j,i',k',l) → i=j Impossible

(i,j',k,l')=(i',k',j,l) → j=k Impossible

(i,j',k,l')=(k',i',j,l) → j=k Impossible

(i,j',k,l')=(j,k',i',l) → i=j Impossible

(i,j',k,l')=(k',j,i',l) → On a alors $i \geq k$ et $i' \geq k'$ avec i'=k et i=k'. On obtient donc $k' \geq k$ et $k' \leq k$ → k=k'=i=i'. Avec j=j' et l=l' on aboutit alors à (i, j, k, l)=(i', j', k', l') ce qui est impossible

(i,j',k,l')=(i',j,l,k') → k=l Impossible

(i,j',k,l')=(j,i',l,k') → i=j Impossible

(i,j',k,l')=(i',l,j,k') → j=k Impossible

(i,j',k,l')=(l,i',j,k') → i=l Impossible

(i,j',k,l')=(j,l,i',k') → i=j Impossible

(i,j',k,l')=(l,j,i',k') → i=l Impossible

(i,j',k,l')=(i',k',l,j) → l=k Impossible

(i,j',k,l')=(k',i',l,j) → l=k Impossible

(i,j',k,l')=(i',l,k',j) → On a $j' \geq l'$ et $j \geq l$ avec j'=l et l'=j → j'=l' et l=j. Avec i=i' et k=k' on retrouve (i, j, k, l)=(i', j', k', l') ce qui impossible

(i,j',k,l')=(l,i',k',j) → i=l Impossible

(i,j',k,l')=(k',l,i',j) → On a alors $i \geq k$ et $i' \geq k'$ avec $i' = k$ et $i = k'$ → i=i'=k=k'. De plus $j \geq l$ et $j' \geq l'$ avec l=j' et l'=j → (i,j,k,l)=(i',j',k',l') ce qui est impossible.

(i,j',k,l')=(l,k',i',j) → l=i Impossible

(i,j',k,l')=(j,l,k',i') → i=j Impossible

(i,j',k,l')=(l,j,k',i') → i=l Impossible

(i,j',k,l')=(l,k',j,i') → i=l Impossible

(i,j',k,l')=(k',j,l,i') → j=k Impossible

(i,j',k,l')=(k',j,l,i') → k=l Impossible

(i,j',k,l')=(j,k',l,i') → i=j Impossible

Ainsi on a démontré que :

$$\text{II.2} \Rightarrow \forall(p, q, r, s, p', q', r', s') \text{ tel que (p, q, r, s)} \neq \text{permutation de (p', q', r', s')}$$

$$g_p + g_q + g_r + gs \neq g_{p'} + g_{q'} + g_{r'} + g_{s'} \rightarrow g_p - g_{q'} + g_r - g_{s'} \neq g_{p'} - g_q + g_{r'} - g_s$$

Ceci étant vrai $\forall(p, q, r, s, p', q', r', s')$ exceptées les permutations qui ne sont de toutes façons pas comprises dans II.1, on a bien l'implication $II.2 \rightarrow II.1$.

Montrons maintenant la réciproque $II.1 \rightarrow II.2$

On peut reformuler les conditions incluses dans la proposition 1 sous la forme : $\forall(i, j, k, l, i', j', k', l')$ $i \geq k$ et $j \geq l$ si $g_i - g_j + g_k - g_l = g_{i'} - g_{j'} + g_{k'} - g_{l'}$. Alors l'un des deux quintuplets (i,j,k,l,i) ou (i',j',k',l',i') a deux termes consécutifs égaux ou ces quintuplets sont égaux.

On suppose alors que

$$g_i + g_j + g_k + g_l = g_{i'} + g_{j'} + g_{k'} + g_{l'}. \qquad \text{(II.3)}$$

On note $E = \{g_{i'}, g_{j'}, g_{k'}, g_{l'}\}$ et $F = \{i', j', k', l'\}$ (il existe naturellement une bijection pour passer d'un ensemble à l'autre).

On va montrer que g_i appartient nécessairement à E en raisonnant par contraposée.

On supposons donc que g_i n'appartient pas à E.

(II.3) implique :

$$g_i - g_{i'} + g_j - g_{j'} = g_{k'} - g_k + g_{l'} - g_l$$

$$g_i - g_{j'} + g_j - g_{k'} = g_{l'} - g_k + g_{i'} - g_l$$
$$g_i - g_{k'} + g_j - g_{l'} = g_{i'} - g_k + g_{j'} - g_l$$
$$g_i - g_{l'} + g_j - g_{i'} = g_{j'} - g_k + g_{k'} - g_l$$

Ainsi d'après l'hypothèse initiale on a:
(i,i',j,j',i) ou (k',k,l,l',k') a deux termes consécutifs égaux ou sont égaux
(i,j',j,k',i) ou (l',k,i',l,l') a deux termes consécutifs égaux ou sont égaux
(i,k',j,l',i) ou (i',k,j',l,i') a deux termes consécutifs égaux ou sont égaux
(i,l',j,i',i) ou (j',k,k',l,j') a deux termes consécutifs égaux ou sont égaux
On peut effectuer le même raisonnement en privilégiant (i,k) et (i,l) comme premier et troisième termes de l'équation de gauche. On aboutit alors à (en notant schématiquement):

$$\left.\begin{array}{l|l} (i,i',j,j',i) & (k,k',l',l,k') \\ (i',j',j,k',i) & (l',k,i',l,l') \\ (i,k',j,l',i) & (i',k,j',l,i') \\ (i,l',j,i',i) & (j',k,k',l,j') \end{array}\right\} série1$$

$$\left.\begin{array}{l|l} (i,i',k,j',i) & (k',j,l',l,k') \\ (i,j',k,k',i) & (l',j,i',l,l') \\ (i,k',k,l',i) & (i',j,j',l,l') \\ (i,l',k,i',i) & (j',j,k',l,j') \end{array}\right\} série2$$

$$\left.\begin{array}{l|l} (i,i',l,j',i) & (k',j,l',k,k') \\ (i,j',l,k',i) & (l',j,i',k,l') \\ (i,k',l,l',i) & (i',j,j',k,i') \\ (i,l',l,i',i) & (j',j,k',k,j') \end{array}\right\} série3$$

On va montrer qu'au moins quatre des quintuplets "de droite" d'une série ont

deux termes consécutifs égaux.

Tout d'abord on élimine la possibilité selon laquelle un quintuplet de droite pourrait être égal à un quintuplet de gauche. On aurait sinon en effet i=k', i=l', i=i', i=j' ce qui imposerait i ∈ F situation contraire à l'hypothèse formulée.

Si l'on suppose qu'il n'est pas possible d'avoir dans la même série tous les quintuplets de droite avec deux termes égaux, il en existe forcément un "à gauche" dans chaque série qui posséde cette propriété. Comme i n'appartient pas à F, on a donc trois égalités avec j, k, l telles que j, k, l appartiennent à F.

En considérant que deux au moins des trois termes j, k, l sont distincts par exemple $j \neq k$ et $k \neq l$ alors comme j ∈ F, k ∈ F et l ∈ F on a par exemple j=j' et k=k'.
Ainsi (II.3) devient :

$$g_i + g_l = g_{i'} + g_{l'}$$

$$g_i - g_{i'} = g_{l'} - g_l$$

$$g_i - g_{i'} + g_i - g_{i'} = g_{l'} - g_l + g_{l'} - g_l$$

Donc soit (i,i',i,i',i), soit (l',l,l',l,l') sont égaux ou ont deux termes consécutifs égaux. Comme i n'appartient pas à F on a forcément l=l' → i=i' ce qui n'est pas cohérent avec l'hypothèse formulée.

Si on suppose au contraire que j=k=l=(par exemple) l'équation II.3 devient :

$$g_i + g_j + g_k = g_{i'} + g_{j'} + g_{k'}$$

$$g_i - g_{i'} + g_k - 0 = g_{j'} - g_j + g_{k'} - 0$$

Soit (i,i',k,0,i) ou (j',j,k',0,j') sont égaux ou ont deux termes consécutifs égaux.
si i=i' → impossible i ∈ F
si i'=k=l' → i'=k'=j'=l'=k → $g_i + g_j = g_{i'} + g_{j'}$ cas déja traité

si i=0 $\rightarrow g_i = g_j = g_k = g_l = 0 = g_{i'}$ impossible

si k=0 $\rightarrow g_{l'} = g_j = g_k = g_l = 0$ d'où $g_i = g_{i'} + g_{j'} + g_{k'}$

$g_i - g_{i'} = g_{j'} + g_{k'}$ $g_i - g_{i'} + 0 - g_{i'} = g_{j'} - g_{i'} + g_{k'} - 0$

$\rightarrow$ étude de la paire (i,i',0,i',i) et (j',i',k',0,i')

Il est clair que le premier quintuplet ne peut avoir deux termes consécutifs égaux (sinon soit i'=0 soit i=i' $\in$ F) et que les deux quintuplets ne peuvent être égaux (sinon i appartiendrait à F). On sait donc que seul le quintuplet "de droite" peut avoir deux termes consécutifs égaux. Etudions chaque cas en détails:

si i'=0 tous les g_p sont nuls ce qui est impossible

si j'=i' $\rightarrow$

$$g_i - g_{i'} + 0 - g_{i'} = g_{k'} - 0$$

$$g_i - g_{i'} = g_{k'} + g_{i'}$$

$$g_i - g_{i'} + g_i - g_{i'} = g_{k'} - 0 + g_i - 0$$

On étudie la paire de quintuplets (i,i',i,i',i) et (k',0,i,0,k'). De même que précédemment on montre que la seule possibilité induite par l'équation précédente est que le quintuplet de droite posséde deux termes consécutifs égaux

- si k'=0 $\rightarrow$

$$g_i - g_{i'} + 0 - g_{i'} = 0$$

$$g_i - g_{i'} + g_i - g_{i'} = g_i$$

$$g_i - g_{i'} + g_i - 0 = g_i - 0 + g_{i'} - 0$$

On considère alors les deux paires (i,i',i,i',i) et (i,0,i',0,i). Les trois cas: les deux quintuplets sont égaux, le premier ou le deuxième quintuplet possède deux termes consécutifs égaux sont impossibles du fait de i et i' différents de 0 et de la non appartenance de i à F.

- si i=0 $\rightarrow$ impossible

si i'=k' alors comme $i' \geq j' \geq k'$ on obtient j'=i' et on se reporte au cas précédent

si i'=0 tous les $g - k$ sont nuls ce qui est impossible

si k'=0 $\rightarrow$

$$g_i = g_{i'} + g_{j'} \; g_i - g_{i'} + g_i - g_{i'} = g_{j'} - 0 + g_{j'} - 0$$

On étudie alors la paire de quintuplets (i,i',i,i',i) et (j',0,j',0,j'). L'égalité des deux ensembles ne peut être effective sinon i appartiendrait à F ce qui empêche également l'égalité i=i' et donc le premier quintuplet ne peut avoir deux termes consécutifs égaux. Le cas du deuxième quintuplet est également réglé fort simplement car si j'=0 alors on obtient d'après les équations $g_i = g_{i'}$ ce qui est impossible.

si j'=j alors on se ramène à un cas déja traité avec $g_i - g_{k'} = g_{i'} - g_k$ (cf j'=i' du cas k=0)

si j=k' on arrive à $g_i - g_{i'} = g_{j'} - g_k$ situation déja étudiée

si k'=0 alors l'=0 et k=0 et on retrouve un cas déja traité

si j'=0 alors k'=0 et on se retrouve avec le cas précédent

Les diverses possibilités étudiées aboutissent toutes à une contradiction ce qui invalide l'hypothèse de départ. Ainsi on a forcément au moins une série de droite (p. 153) qui ne contient que des quintuplets ayant au minimum deux termes consécutifs égaux. On suppose pour fixer les idées que cette propriété est vérifiée pour la série 1.

On rappelle ces quintuplets ci dessous :

(k',k,l',l,k')

(l',k,i',l,l')

(i',k,j',l,i')

(j',k,k',l,j')

On dénombre alors trois possibilités : soit k, soit l, soit les deux, appartiennent à F.

Si k et l appartiennent à F :

si $k \neq l$ alors II.3 devient $g_i + g_j = g_{i'} + g_{j'}$ qui est un cas déja traité aboutissant à une contradiction

si k=l alors forcément au moins deux termes de E sont égaux (traduction de k=l et de k, l appartiennent à F sur les quatres quintuplets) et II.3 devient également $g_i + g_j = g_{i'} + g_{j'}$.

Si seul l appartient à F : on a nécessairement soit l=l' et l=j' soit l=k' et l=i'. si l=l' et l=j', II.3 devient $g_i + g_j + g_k = g_{i'} + g_{j'} + g_{j'}$

$g_i - g_{i'} + g_j - 0 = g_{j'} - g_k + g_{j'} - 0$

On se retrouve également ici avec l'étude de deux ensembles (i,i',j,0,i) et (j',k,j',0,j'). Comme cela a déja été établi auparavant ces deux ensembles ne peuvent être identiques.

si j=i' $\rightarrow$ $g_i + g_k = g_{i'} + g_{j'}$ cas déja traité

si i=0 situation impossible

si j=0 $\rightarrow$ $g_k = g_l = g_{l'} = g_{j'} = 0 \rightarrow g_i = g_{i'}$

si k=j' $\rightarrow$ cas connu ($g_i + g_j = g_{j'} + g_{i'}$)

si j'=0 $\rightarrow$ $g_i + g_j + g_k = g_{i'}$

$g_{i'} - g_i + g_{i'} - g_j = g_{i'} - 0 + g_k - 0$

Une nouvelle fois étude de la paire de quintuplets (i',i,i',j,i') et (i',0,k,0,i')

ces deux ensembles sont non égaux sinon i=0

si j=i' $\rightarrow$ $g_k = -g_{i'} \rightarrow g_i = 0$

si i=i' $\rightarrow$ impossible

si k=0 $\rightarrow$ $g_{i'} - g_i = g_j$

$g_{i'} - g_i + g_{i'} - g_i = g_j - 0 + g_j - 0$

Nouvelle étude des deux quintuplets (i',i,i',i,i') et (j,0,j,0,j)

Ils sont non égaux car i n'appartient pas à F

de même $i \neq i'$

j=0 $\rightarrow$ $g_i = g_{i'}$ ce qui est impossible

si l=k' et l=i' comme $g_i \geq g_j \geq g_k \geq g_l = g_{i'} \geq g_{j'} \geq g_{k'} \geq g_{l'}$ et $g_i + g_j + g_k + g_l = g_{i'} + g_{j'} + g_{k'} + g_{l'}$ la seule possibilité est

$g_i = g_j = g_k = g_l = g_{i'} = g_{j'} = g_{k'} = g_{l'}$ situation impossible

si seul k appartient à F, on a soit k=l' et k=j' soit k=k' et k=i'

si k=k' et k=i' $g_i+g_j+g_k+g_l = g_{i'}+g_{j'}+g_{k'}+g_{l'}$ devient $g_i+g_j+g_l = g_{i'}+g_{j'}+g_{l'}$

$g_i - g_{i'} + g_l = g_{i'} - g_j + g_{l'} - 0$ On aboutit alors une nouvelle fois à l'étude de deux quintuplets (i,i',l,0,i) et (i',j,l',0,i').

Pour les mêmes raisons que précédemment ces deux ensembles ne sont pas égaux.

si i'=l alors du fait de $g_i \geq g_j \geq g_k \geq g_l = g_{i'} \geq g_{j'} \geq g_{k'} \geq g_{l'} \rightarrow$ tous égaux $\rightarrow$ impossible si i=0 $\rightarrow$ problème

si i'=j $\rightarrow g_i + g_l = g_{i'} + g_{l'}$ cas déja traité

si l'=j $\rightarrow g_i + g_l = g_{i'} + g_{i'}$ cas déja traité

si l=0 $\rightarrow g_i + g_j = g_{i'} + g_{i'} + g_{l'}$ avec $g_i \geq g_j \geq g_k = g_{i'} \geq g_{i'} \geq g_{l'} \rightarrow$ tous égaux, cas impossible

si l'=0 $\rightarrow g_i + g_j + g_l = g_{i'} + g_{i'}$

avec $g_i \geq (g_j) \geq (g_k) = g_{i'} \rightarrow g_l = 0$ cas précédent

si k=l' et k=j' $g_i + g_j + g_l = g_{i'} + g_{j'} + g_{j'}$

$g_i - g_{i'} + g_j - 0 = g_{j'} - g_l + g_{j'} - 0$

Etude de la paire: (i,i',j,0,i) et (j',l,j',0,j')

si i=i' $\rightarrow$ impossible

si i'=j $\rightarrow g_i + g_l = g_{j'} + g_{l'}$ cas classique identique à ceux déja traités

si j=0 $\rightarrow g_k = 0 \rightarrow g_{j'} = g_{l'} = g_{k'} = 0 \rightarrow g_i = g_{i'}$ cas impossible

si j'=l $\rightarrow g_i + g_j = g_{i'} + g_{j'}$cas déja étudié

si j'=0 $\rightarrow g_i + g_j = g_{i'}$ $g_i - 0 + g_i - 0 = g_{i'} - g_j + g_{i'} - g_j$ Etude de la paire (i,0,i,0,i) (i',j,i',j,i')

si i=0 $\rightarrow$ problème

si i'=j $\rightarrow g_i = 0$ impossible

Ainsi tous les cas possibles avec l'hypothèse de départ aboutissent à une contradiction. Ceci implique que notre point de départ était erroné et donc que g_i ap-

partient à E. On applique le même raisonnement pour prouver que g_j et g_l appartiennent à E.

Ainsi l'on a prouvé l'implication : $1 \rightarrow 2$ (on est forcément en présence d'une permutation s'il y a égalité).

L' équivalence est donc bien établie.

Annexe III

Adaptation de l'algorithme GARSP

Il est envisageable d'aborder la généralisation du problème des règles de Golomb sous un angle différent. Au lieu d'essayer de se raccrocher aux résultats proprement dits (séquences établies 4.2.3), il peut être intéressant de voir s'il n'est pas possible d'adapter à notre problème les différents algorithmes qui ont permis la génération de ces ensembles. On s'intéressera plus particulièrement à la méthode GARSP.

L'étude présentée ici est basée sur le travail de maîtrise réalisé par Sien Cuong Hoa sous la direction de Mme Jaumard au sujet des règles de Golomb. On reprendra ici les mêmes notations définies dans le cadre des structures *bitmaps* ([48]).

Les structures *bitmaps* sont des segments de bits servant à représenter nos ensembles de générateurs. Chaque entité de ce segment prend la valeur 0 ou 1 selon si l'élément correspondant à cette position est bien présent dans notre ensemble. Par exemple, l'ensemble {0,1,5} est dénoté par 110001.

On définit :
$DIST(m)$ une structure *bitmap* contenant toutes les différences engendrées par l'ensemble en cours contenant m éléments (distances interdites).
$LENGTH$ est un vecteur contenant les positions des éléments de l'ensemble considéré (ex : $LENGTH(1) = 0$ pour l'ensemble {0,1,5}).

Un des points délicats de cette généralisation consiste à mettre en équation la condition d'évolution permettant de passer de $LENGTH(m)$ à $LENGTH(m+1)$ en tenant compte des distances interdites ($DIST(m)$). Le résultat obtenu est le

suivant :

$$LENGTH(m+1) \neq DIST(m) + \underbrace{(LENGTH(i) + LENGTH(k) - LENGTH(j))}_{\textit{avec soit i, soit k, soit j=m et si j=m et i(k)=m alors k(i)=m}} \quad \text{(III.1)}$$

$DIST(m)$ est naturellement calculé à partir de toutes les combinaisons des entiers contenu dans $LENGTH$ selon la condition de double orthogonalité (I.2).

Illustrons cela par un exemple :

Partons d'un ensemble doublement orthogonal : $LENGTH = [0\ 1\ 5]$. Le calcul des distances interdites donne : $DIST(2) = [-10\ -9\ -8\ -6\ -3\ -2\ 2\ 3\ 6\ 8\ 9\ 10]$. Calculons les expressions : $E(i,k,j) = LENGTH(i)+LENGTH(k)-LENGTH(j)$ avec les conditions précédemment évoquées :
$E(2,0,0) = 5 + 0 - 0 = 5$, $E(2,1,0) = 5 + 1 - 0 = 6$, $E(2,2,0) = 5 + 5 + 0 = 10$, $E(2,2,1) = 5 + 5 - 1 = 9$, $E(2,0,1) = 5 + 0 - 1 = 4$, $E(0,0,2) = 0 + 0 - 5 = -5$, $E(0,1,2) = 0 + 1 - 5 = -4$, $E(1,1,2) = 1 + 1 - 5 = -3$.
En additionnant $DIST(2)(p)$ et $E(i,k,j)$ pour toutes les combinaisons, on s'aperçoit que toutes les valeurs entières de 1 à 20 sont atteintes. Le plus petit entier que peut prendre $LENGTH(3)$ est donc 21. On obtient alors l'ensemble doublement orthogonal $\{0, 1, 5, 21\}$.

Annexe IV

Étude temporelle des codes doublement orthogonaux au sens large de taux $\frac{1}{2}$ pour un nombre d'éléments J variant de 6 à 10

Note: par convention ' représentera les minutes et " les secondes.

J=6 (limite de temps: 1h30)

Tableau IV.1 - *Évolution de la long. moyenne en fonction du temps pour J=6 (100)*

longueur	135	115	109	105	103	100
temps moyen requis	inst.	$\leq$ 10"	$\leq$ 30"	$\leq$ 3'	$\leq$ 10'	$\geq$ 48'

J=7 (limite de temps: 3h)

Tableau IV.2 - *Évolution de la long. moyenne en fonction du temps pour J=7 (222)*

l.	330	275	270	255	250	246	240	229
t.	inst.	$\leq$ 20"	$\leq$ 1'30"	$\leq$ 3'	$\leq$ 5'	$\leq$ 10'	$\leq$ 1h	$\geq$ 40'

J=8 (limite de temps: 14h)

Tableau IV.3 - *Évolution de la long. moyenne en fonction du temps pour J=8 (459)*

longueur	690	580	550	545	540	525	515	505
temps requis	inst.	$\leq$ 15'	$\leq$ 40'	$\leq$ 1h30'	$\leq$ 3h	$\leq$ 6h	$\geq$ 7h	$\geq$ 9h

J=9 (limite de temps 18h)

Tableau IV.4 - *Évolution de la long. moyenne en fonction du temps pour J=9 (912)*

longueur	1420	1340	1300	1275	1190	1150	1080
temps requis	inst.	$\leq$ 1"	$\leq$ 1h	$\leq$ 1h30'	$\leq$ 6h	$\leq$ 8h	$\geq$ 9h

J=10 (limite de temps 24h)

Tableau IV.5 - *Évolution de la long. moyenne en fonction du temps pour J=10 (1698)*

l.	3500	2995	2700	2550	2400	2320	2240	2200	2050	2010
t.	inst.	$\leq$ 2"	$\leq$ 6"	$\leq$ 8'	$\leq$ 10'	$\leq$ 15'	$\leq$ 1h30'	$\leq$ 2h	$\geq$ 12h	$\geq$ 24h

Annexe V

Codes doublement orthogonaux au sens strict de taux $\frac{J}{2J}$ pour J variant de 2 à 14

J=2

Tableau V.1 - *Générateurs à longueur réduite d'un code d.o., sens strict, taux* $\frac{2}{4}$

$$\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

J=3

Tableau V.2 - *Générateurs à longueur réduite d'un code d.o., sens strict, taux* $\frac{3}{6}$

$$\begin{pmatrix} \underline{0} & 0 & \underline{5} \\ 1 & 3 & 0 \\ \underline{5} & 2 & \underline{0} \end{pmatrix}$$

J=4

Tableau V.3 - *Générateurs à longueur réduite d'un code d.o., sens strict, taux* $\frac{4}{8}$

$$\begin{pmatrix} 13 & \underline{0} & 24 & \underline{25} \\ 0 & 1 & 0 & 24 \\ 8 & \underline{25} & 14 & \underline{0} \\ 15 & 3 & 23 & 0 \end{pmatrix}$$

J=5

Tableau V.4 - *Générateurs à longueur réduite d'un code d.o., sens strict, taux* $\frac{5}{10}$

$$\begin{pmatrix} \underline{98} & 0 & 0 & \underline{0} & 17 \\ 4 & 12 & 17 & 0 & 92 \\ 0 & 90 & 35 & 26 & 5 \\ \underline{0} & 24 & 28 & \underline{98} & 0 \\ 0 & 32 & 66 & 22 & 17 \end{pmatrix}$$

J=6

Tableau V.5 - *Générateurs à longueur réduite d'un code d.o., sens strict, taux* $\frac{6}{12}$

$$\begin{pmatrix} 110 & 50 & 0 & 301 & 228 & 67 \\ 311 & 42 & 5 & 0 & 79 & 154 \\ 94 & 71 & \underline{0} & 0 & 82 & \underline{323} \\ 0 & 25 & \underline{323} & 18 & 0 & \underline{0} \\ 46 & 0 & 6 & 300 & 36 & 20 \\ 90 & 232 & 0 & 14 & 120 & 69 \end{pmatrix}$$

J=7

Tableau V.6 - *Générateurs à longueur réduite d'un code d.o., sens strict, taux* $\frac{7}{14}$

$$\begin{pmatrix} 173 & 67 & 220 & 0 & 362 & 179 & 89 \\ 126 & 96 & 109 & 0 & 137 & 699 & 20 \\ 0 & 875 & 3 & 262 & 0 & 0 & 111 \\ 16 & 192 & \underline{0} & \underline{879} & 25 & 40 & 0 \\ 270 & 59 & \underline{879} & \underline{0} & 172 & 519 & 61 \\ 565 & 0 & 93 & 79 & 145 & 111 & 878 \\ 219 & 77 & 387 & 0 & 860 & 177 & 64 \end{pmatrix}$$

J=8

Tableau V.7 – *Générateurs à longueur réduite d'un code d.o., sens strict, taux* $\frac{8}{16}$

$$\begin{pmatrix} 584 & 152 & 1034 & 94 & 1292 & 611 & 0 & 550 \\ 547 & 2003 & 2023 & 0 & 302 & 404 & 558 & 409 \\ 1084 & 99 & 486 & 49 & 383 & 778 & 0 & 512 \\ 1660 & 365 & 424 & 0 & 323 & 433 & 2068 & 583 \\ 307 & 0 & 295 & 2138 & 228 & 1180 & 0 & 305 \\ 555 & 173 & 621 & 95 & 434 & 1942 & 0 & 903 \\ 0 & 706 & 0 & 0 & 0 & 0 & \underline{2252} & \underline{0} \\ 548 & 147 & 736 & 821 & 489 & 542 & \underline{0} & \underline{2252} \end{pmatrix}$$

J=9

Tableau V.8 – *Générateurs à longueur réduite d'un code d.o., sens strict, taux* $\frac{9}{18}$

$$\begin{pmatrix} 0 & 119 & \underline{0} & \underline{4804} & 48 & 47 & 2499 & 239 & 952 \\ 2001 & 438 & 2676 & 0 & 1038 & 453 & 442 & 321 & 615 \\ 397 & 782 & 1116 & 0 & 3938 & 529 & 456 & 266 & 403 \\ 715 & 203 & 153 & \underline{0} & 232 & 208 & \underline{4803} & 1360 & 175 \\ 2967 & 0 & \underline{4803} & 201 & 0 & 0 & \underline{0} & 0 & 0 \\ 366 & 3124 & 421 & 0 & 580 & 2285 & 1217 & 238 & 377 \\ 385 & 465 & 379 & 0 & 1842 & 875 & 562 & 258 & 4035 \\ 493 & 1634 & 691 & 0 & 447 & 4597 & 440 & 287 & 402 \\ 298 & 326 & 277 & 0 & 329 & 1227 & 717 & 4221 & 2455 \end{pmatrix}$$

J=10

Tableau V.9 – *Générateurs à longueur réduite d'un code d.o., sens strict, taux* $\frac{10}{20}$

$$\begin{pmatrix}
0 & 3654 & 1732 & 1836 & 1638 & 2226 & 3297 & 1683 & 1599 & 7778 \\
0 & 591 & 1636 & 1570 & 2750 & 7308 & 1620 & 1630 & 1684 & 3409 \\
0 & 955 & 5693 & 2986 & 1515 & 1562 & 7887 & 1566 & 1490 & 2367 \\
1295 & 9709 & 118 & 949 & 0 & 0 & 309 & 0 & 6979 & 2 \\
\underline{9778} & 4773 & 0 & 0 & 287 & 4 & \underline{0} & 2445 & 793 & 0 \\
\underline{0} & 131 & 2884 & 5474 & 1136 & 1167 & \underline{9778} & 1451 & 1082 & 1202 \\
0 & 660 & 1696 & 8757 & 1888 & 3839 & 2281 & 1788 & 1633 & 1694 \\
751 & 0 & 1479 & 958 & 4738 & 1120 & 1005 & 8581 & 932 & 1009 \\
0 & 834 & 1699 & 1641 & 1655 & 1704 & 1722 & 2442 & 4149 & 5270 \\
3575 & 1 & 9730 & 20 & 1975 & 92 & 196 & 115 & 0 & 437
\end{pmatrix}$$

J=11

Tableau V.10 – *Générateurs à longueur réduite d'un code d.o., sens strict, taux* $\frac{11}{22}$

$$\begin{pmatrix}
4554 & 540 & 8515 & 1589 & 2407 & 681 & \underline{18305} & \underline{0} & 94 & 170 & 488 \\
308 & 2302 & 431 & 436 & 21 & 7291 & 428 & 12595 & 0 & 0 & 646 \\
9706 & 0 & 0 & 0 & \underline{18306} & 0 & \underline{0} & 2759 & 153 & 1591 & 0 \\
0 & 184 & 159 & 498 & 6549 & 813 & 150 & 339 & 4407 & 14798 & 61 \\
1048 & 4579 & 16104 & 1193 & 0 & 12007 & 1385 & 131 & 712 & 1893 & 1102 \\
1936 & 1246 & 1414 & 1282 & 0 & 1305 & 3050 & 5648 & 811 & 1254 & 17267 \\
1241 & 1694 & 4307 & 3761 & \underline{0} & 2452 & 1311 & \underline{18306} & 793 & 845 & 10135 \\
1282 & 13299 & 1279 & 7072 & 0 & 1523 & 1298 & 201 & 2279 & 849 & 1199 \\
14172 & 789 & 876 & 819 & 1892 & 4978 & 1560 & 0 & 7502 & 565 & 729 \\
935 & 1054 & 2541 & 1122 & 0 & 1068 & 4770 & 13 & 607 & 6957 & 1562 \\
2442 & 1231 & 1749 & 13009 & 0 & 1248 & 9067 & 245 & 1002 & 835 & 3972
\end{pmatrix}$$

J=12

Tableau III.11: *générateurs à longueur réduite d'un code de taux* $\frac{12}{24}$

376	17827	0	44	10	13421	0	9740	5085	214	1063	2496
3452	0	2090	30846	1496	1305	16545	5056	1328	10300	1313	1350
6077	0	748	742	810	738	33161	1081	2803	22047	753	11592
21651	0	1322	1308	1307	3969	2071	15303	1306	7484	1713	1444
2308	0	1387	1311	4704	7280	1324	1304	1588	1312	18567	27598
1357	0	1309	1315	25382	1707	4438	1512	13442	1339	7124	3022
8645	33160	12909	315	38	0	0	4341	129	1950	1493	0
1282	0	5906	1912	2837	1273	1279	27834	1278	1364	1454	8911
153	0	36	2928	516	53	37	27	16454	37	10626	40
0	5839	1829	3986	0	23020	124	0	0	550	27	2
0	1327	20658	0	8161	56	304	1183	798	0	0	8
1209	0	1476	19522	1211	2598	8794	1265	1219	5730	30282	2088

J=13

Tableau III.12: *générateurs à longueur réduite d'un code de taux* $\frac{13}{26}$

22130	0	1592	1883	12941	2522	29211	345	2373	8980	2713	4383	2684
0	52	7362	205	0	15033	0	31175	0	0	0	51999	0
47237	52000	0	7171	2077	0	412	10009	22	20978	429	0	3586
7441	0	15851	3747	44036	1320	1919	21922	1439	1911	2201	1517	4059
5233	341	715	1000	1704	1196	13438	0	32131	1861	2012	6531	1802
1412	15649	74	0	695	162	818	916	4290	37882	879	404	8871
1100	4390	2880	182	808	285	937	0	41814	1447	11954	28252	913
2810	0	45475	7868	33906	12070	5274	405	2270	2920	2747	3291	3072
4035	0	3616	38093	26256	2101	7016	457	2291	16391	2752	2595	3472
40998	0	24239	2125	7262	26463	2728	469	4146	3597	9959	2313	2695
15034	0	1671	2354	3762	3619	2759	380	20591	51999	6305	10271	2867
2756	0	1630	18977	3208	6491	2778	381	2278	4364	35799	2415	16224
2765	0	1801	1923	2671	2099	3066	4420	11364	2746	3652	21611	31026

J=14

Tableau III.13: *générateurs à longueur réduite d'un code de taux* $\frac{14}{28}$

19818	24822	0	2823	8383	6052	35277	6090	2575	4392	6488	71717	3850	6493
12109	6358	0	3125	7330	1920	3907	12345	2412	4421	67197	23135	2817	35442
58905	9704	0	2830	6878	1921	9978	6008	32976	5607	6576	6524	15353	8797
7210	4338	52854	4846	4592	677	0	18552	470	2470	4558	14536	47913	4635
6094	5181	15557	80278	4797	245	0	7265	841	50718	11851	39415	1116	4800
4408	4256	0	18638	14978	0	80901	3973	392	29925	4813	9938	802	5234
0	0	80901	26907	0	2564	8959	0	45444	0	0	0	0	0
3579	3467	22558	0	3797	12873	40517	75651	0	5102	4743	3650	1	12266
23720	80901	3074	993	28752	89	0	49716	606	3025	4659	5617	2717	11454
6410	51749	0	11763	6504	3663	1703	5992	7041	4599	49118	7031	2883	28190
5918	5777	0	2370	6003	39814	5568	6968	11448	3903	17692	5985	28632	6265
6246	33099	0	4277	12415	59271	2130	5833	18483	4234	6380	8780	2670	80092
66252	14098	0	2784	43003	1867	1645	27958	3550	7055	9563	6631	3121	6444
4886	5857	3460	1933	69395	170	0	4239	655	14232	24959	4732	36528	4761

Annexe VI

Codes en blocs doublement orthogonaux de taux $\frac{J}{K+J}$ avec $2 \leq J,\ K \leq 10$

Tableau VI.1 – *Générateurs d'un code en blocs d.o. de taux* $\frac{2}{4}$

0	0
0	1

Tableau VI.2 – *Générateurs d'un code en blocs d.o. de taux* $\frac{2}{5}$

0	$\underline{0}$	$\underline{2}$
0	$\underline{1}$	$\underline{0}$

Tableau VI.3 – *Générateurs d'un code en blocs d.o. de taux* $\frac{2}{6}$

0	$\underline{0}$	2	$\underline{3}$
1	$\underline{3}$	0	$\underline{0}$

Tableau VI.4 – *Générateurs d'un code en blocs d.o. de taux* $\frac{2}{7}$

0	0	$\underline{0}$	4	$\underline{6}$
1	2	$\underline{5}$	0	$\underline{0}$

Tableau VI.5 – *Générateurs d'un code en blocs d.o. de taux* $\frac{2}{8}$

1	6	0	8	$\underline{0}$	$\underline{9}$
0	0	3	0	$\underline{9}$	$\underline{0}$

Tableau VI.6 – *Générateurs d'un code en blocs d.o. de taux* $\frac{2}{9}$

0	6	$\underline{0}$	0	0	0	$\underline{15}$
13	0	$\underline{15}$	8	12	2	$\underline{0}$

Tableau VI.7 – *Générateurs d'un code en blocs d.o. de taux* $\frac{2}{10}$

10	0	0	3	$\underline{0}$	13	14	$\underline{19}$
0	5	17	0	$\underline{19}$	0	0	$\underline{0}$

Tableau VI.8 – *Générateurs d'un code en blocs d.o. de taux* $\frac{2}{11}$

0	28	$\underline{0}$	0	$\underline{26}$	0	14	0	0
12	0	$\underline{27}$	8	$\underline{0}$	3	0	9	19

Tableau VI.9 – *Générateurs d'un code en blocs d.o. de taux* $\frac{2}{12}$

0	0	0	0	0	10	$\underline{0}$	2	22	$\underline{37}$
7	3	35	4	14	0	$\underline{37}$	0	0	$\underline{0}$

Tableau VI.10 – *Générateurs d'un code en blocs d.o. de taux* $\frac{3}{5}$

0	0
$\underline{0}$	$\underline{1}$
$\underline{2}$	$\underline{0}$

Tableau VI.11 – *Générateurs d'un code en blocs d.o. de taux* $\frac{3}{6}$

$\underline{0}$	0	$\underline{5}$
1	3	0
$\underline{5}$	2	$\underline{0}$

Tableau VI.12 – *Générateurs d'un code en blocs d.o. de taux* $\frac{3}{7}$

6	11	4	0
0	0	$\underline{11}$	$\underline{0}$
1	3	$\underline{0}$	$\underline{11}$

Tableau VI.13 - *Générateurs d'un code en blocs d.o. de taux* $\frac{3}{8}$

19	0	14	17	0
0	$\underline{0}$	5	0	$\underline{21}$
8	$\underline{20}$	0	2	$\underline{0}$

Tableau VI.14 - *Générateurs d'un code en blocs d.o. de taux* $\frac{3}{9}$

26	22	0	28	23	35
$\underline{0}$	0	35	0	$\underline{36}$	0
$\underline{36}$	18	22	8	$\underline{0}$	7

Tableau VI.15 - *Générateurs d'un code en blocs d.o. de taux* $\frac{3}{10}$

22	23	47	51	0	13	38
0	48	0	0	$\underline{0}$	$\underline{57}$	0
41	0	10	8	$\underline{56}$	$\underline{0}$	7

Tableau VI.16 - *Générateurs d'un code en blocs d.o. de taux* $\frac{3}{11}$

14	$\underline{0}$	66	56	0	$\underline{88}$	75	49
0	$\underline{88}$	0	0	70	$\underline{0}$	0	69
12	44	56	60	61	55	53	0

Tableau VI.17 - *Générateurs d'un code en blocs d.o. de taux* $\frac{3}{12}$

0	0	1	122	0	0	0	74	3
50	101	$\underline{124}$	0	41	27	7	0	$\underline{0}$
74	63	$\underline{0}$	12	57	36	18	40	$\underline{123}$

Tableau VI.18 - *Générateurs d'un code en blocs d.o. de taux* $\frac{3}{13}$

$\underline{0}$	39	45	$\underline{178}$	46	59	55	45	0	47
83	0	65	18	0	43	173	0	11	91
$\underline{178}$	61	0	$\underline{0}$	19	0	0	3	167	0

Tableau VI.19 – *Générateurs d'un code en blocs d.o. de taux* $\frac{4}{6}$

0	1
$\underline{0}$	$\underline{3}$
2	0
$\underline{3}$	$\underline{0}$

Tableau VI.20 – *Générateurs d'un code en blocs d.o. de taux* $\frac{4}{7}$

6	0	1
11	0	3
4	$\underline{11}$	$\underline{0}$
0	$\underline{0}$	$\underline{11}$

Tableau VI.21 – *Générateurs d'un code en blocs d.o. de taux* $\frac{4}{8}$

13	$\underline{0}$	24	$\underline{25}$
0	1	0	24
8	$\underline{25}$	14	$\underline{0}$
15	3	23	0

Tableau VI.22 – *Générateurs d'un code en blocs d.o. de taux* $\frac{4}{9}$

1	6	$\underline{54}$	10	$\underline{0}$
21	38	0	0	28
35	37	$\underline{0}$	34	$\underline{53}$
0	0	27	49	0

Tableau VI.23 – *Générateurs d'un code en blocs d.o. de taux* $\frac{4}{10}$

63	85	84	83	$\underline{86}$	$\underline{0}$
83	0	46	0	$\underline{0}$	$\underline{86}$
0	7	0	62	16	80
13	65	25	46	31	0

Tableau VI.24 - *Générateurs d'un code en blocs d.o. de taux* $\frac{4}{11}$

0	146	149	57	104	126	136
149	0	0	7	39	0	131
9	144	142	0	76	110	66
0	113	72	148	0	41	0

Tableau VI.25 - *Générateurs d'un code en blocs d.o. de taux* $\frac{4}{12}$

43	128	134	204	46	152	108	0
239	0	0	0	87	0	40	200
27	87	135	67	0	238	9	192
0	74	81	46	239	83	0	3

Tableau VI.26 - *Générateurs d'un code en blocs d.o. de taux* $\frac{4}{13}$

7	16	335	45	342	18	151	0	0
160	69	0	0	0	126	46	41	33
0	0	42	203	0	0	0	45	343
42	50	0	3	24	82	55	270	129

Tableau VI.27 - *Générateurs d'un code en blocs d.o. de taux* $\frac{4}{14}$

0	449	317	467	0	0	0	140	0	0
214	0	0	36	259	213	125	39	406	350
171	40	73	0	193	225	73	0	99	467
233	0	34	344	448	221	164	41	208	217

Tableau VI.28 - *Générateurs d'un code en blocs d.o. de taux* $\frac{5}{7}$

0	1
0	2
0	5
4	0
6	0

Tableau VI.29 – *Générateurs d'un code en blocs d.o. de taux* $\frac{5}{8}$

19	0	8
0	$\underline{0}$	$\underline{20}$
14	5	0
17	0	2
0	$\underline{21}$	$\underline{0}$

Tableau VI.30 – *Générateurs d'un code en blocs d.o. de taux* $\frac{5}{9}$

1	21	35	0
6	38	37	0
$\underline{54}$	0	$\underline{0}$	27
10	0	34	49
$\underline{0}$	28	$\underline{53}$	0

Tableau VI.31 – *Générateurs d'un code en blocs d.o. de taux* $\frac{5}{10}$

28	26	0	14	0
41	21	$\underline{103}$	58	$\underline{0}$
92	33	$\underline{0}$	44	$\underline{104}$
0	0	103	0	61
29	103	0	24	4

Tableau VI.32 – *Générateurs d'un code en blocs d.o. de taux* $\frac{5}{11}$

136	0	135	194	90	48
192	$\underline{0}$	122	75	$\underline{221}$	17
0	216	0	1	221	80
102	0	98	49	48	0
66	$\underline{221}$	81	0	$\underline{0}$	4

Tableau VI.33 – *Générateurs d'un code en blocs d.o. de taux* $\frac{5}{12}$

131	146	206	175	0	158	293
0	81	14	2	185	0	0
349	128	227	137	0	142	138
11	0	0	0	353	68	0
0	353	148	250	0	17	8

Tableau VI.34 – *Générateurs d'un code en blocs d.o. de taux* $\frac{5}{13}$

0	322	157	188	0	134	122	16
510	107	175	310	74	138	97	0
133	263	259	0	132	272	515	173
0	0	0	516	117	0	0	510
77	193	403	0	105	306	182	80

Tableau VI.35 – *Générateurs d'un code en blocs d.o. de taux* $\frac{5}{14}$

0	0	0	784	0	0	0	769	109
166	743	151	0	744	265	196	70	0
218	117	145	0	353	784	159	539	68
247	43	212	54	298	551	224	0	57
335	19	702	0	174	185	179	333	2

Tableau VI.36 – *Générateurs d'un code en blocs d.o. de taux* $\frac{5}{15}$

0	0	0	87	0	0	1135	39	387	132
299	320	679	333	418	945	0	296	198	208
0	146	5	0	60	210	552	0	193	1073
142	107	977	106	123	90	192	1003	0	3
1135	794	172	310	99	92	0	97	0	0

Tableau VI.37 – *Générateurs d'un code en blocs d.o. de taux* $\frac{6}{8}$

1	0
6	0
0	3
8	0
0	9
9	0

Tableau VI.38 – *Générateurs d'un code en blocs d.o. de taux* $\frac{6}{9}$

26	0	36
22	0	18
0	35	22
28	0	8
23	36	0
35	0	7

Tableau VI.39 – *Générateurs d'un code en blocs d.o. de taux* $\frac{6}{10}$

63	83	0	13
85	0	7	65
84	46	0	25
83	0	62	46
86	0	16	31
0	86	80	0

Tableau VI.40 – *Générateurs d'un code en blocs d.o. de taux* $\frac{6}{11}$

136	192	0	102	66
0	$\underline{0}$	216	0	$\underline{221}$
135	122	0	98	81
194	75	1	49	0
90	$\underline{221}$	221	48	$\underline{0}$
48	17	80	0	4

Tableau VI.41 – *Générateurs d'un code en blocs d.o. de taux* $\frac{6}{12}$

283	308	289	0	240	302
120	0	313	232	0	111
339	154	243	0	121	240
228	317	156	0	27	158
0	209	$\underline{0}$	$\underline{363}$	314	0
271	165	$\underline{363}$	$\underline{0}$	187	272

Tableau VI.42 – *Générateurs d'un code en blocs d.o. de taux* $\frac{6}{13}$

0	23	26	$\underline{0}$	441	12	$\underline{664}$
168	16	107	0	15	51	105
123	168	134	128	283	136	0
134	254	552	629	439	117	0
35	0	0	$\underline{664}$	0	$\underline{0}$	277
361	130	169	116	125	$\underline{664}$	$\underline{0}$

Tableau VI.43 – *Générateurs d'un code en blocs d.o. de taux* $\frac{6}{14}$

0	$\underline{1053}$	$\underline{0}$	0	403	0	0	514
269	0	152	248	303	256	709	338
544	269	625	587	317	941	526	0
848	$\underline{0}$	$\underline{1053}$	203	0	210	450	1047
681	558	679	1047	358	541	521	0
532	268	463	496	433	558	528	0

Tableau VI.44 – *Générateurs d'un code en blocs d.o. de taux* $\frac{6}{15}$

0	1280	223	352	68	9	27	341	12
78	365	81	342	0	0	626	394	1561
1186	368	117	1170	19	24	0	231	611
11	589	162	701	135	33	0	230	16
0	981	87	358	322	132	0	256	0
289	0	0	0	1153	1560	125	0	0

Tableau VI.45 – *Générateurs d'un code en blocs d.o. de taux* $\frac{6}{16}$

0	0	780	1362	0	0	85	395	2023	776
311	775	2090	0	206	81	79	16	2	19
168	0	1445	2089	22	90	0	0	168	0
1916	268	437	668	527	610	224	269	0	414
453	270	30	244	440	341	745	1783	0	574
520	286	0	253	822	346	2089	1211	8	299

Tableau VI.46 – *Générateurs d'un code en blocs d.o. de taux* $\frac{7}{9}$

0	13
6	0
0	15
0	8
0	12
0	2
15	0

Tableau VI.47 – *Générateurs d'un code en blocs d.o. de taux* $\frac{7}{10}$

22	0	41
23	48	0
27	0	10
51	0	8
0	0	56
13	57	0
38	0	7

Tableau VI.48 – *Générateurs d'un code en blocs d.o. de taux* $\frac{7}{11}$

0	149	9	0
146	0	144	113
149	0	142	72
57	7	0	148
104	39	76	0
126	0	110	41
136	131	66	0

Tableau VI.49 – *Générateurs d'un code en blocs d.o. de taux* $\frac{7}{12}$

131	0	349	11	0
146	81	128	0	353
206	14	227	0	148
175	2	137	0	250
0	185	0	353	0
158	0	142	68	17
293	0	138	0	8

Tableau VI.50 – *Générateurs d'un code en blocs d.o. de taux* $\frac{7}{13}$

0	168	123	134	35	361
23	16	168	254	0	130
26	107	134	552	0	169
0	0	128	629	664	116
441	15	283	439	0	125
12	51	136	117	0	664
664	105	0	0	277	0

Tableau VI.51 – *Générateurs d'un code en blocs d.o. de taux* $\frac{7}{14}$

0	98	273	970	406	202	224
0	37	198	199	12	131	189
1101	697	77	315	0	527	105
67	0	177	183	38	103	1043
0	167	0	0	109	1102	18
0	93	306	289	733	278	213
24	92	512	130	1101	0	0

Tableau VI.52 – *Générateurs d'un code en blocs d.o. de taux* $\frac{7}{15}$

612	553	1530	216	30	1856	0	798
281	218	846	0	91	0	816	298
0	0	0	737	1575	0	1856	0
725	695	792	1839	116	407	0	714
512	441	936	106	1100	1083	0	804
533	483	465	171	0	338	230	1261
687	621	597	549	1576	576	0	702

Tableau VI.53 - *Générateurs d'un code en blocs d.o. de taux* $\frac{7}{16}$

100	344	407	0	369	323	309	406	292
0	1972	373	11	2487	1291	155	811	156
0	451	0	2633	0	0	716	0	24
126	565	855	0	371	382	510	440	2634
534	0	2633	1395	1385	168	0	332	0
110	277	402	0	1079	703	290	2455	450
199	1521	457	0	372	2328	681	413	286

Tableau VI.54 - *Générateurs d'un code en blocs d.o. de taux* $\frac{7}{17}$

1292	723	484	3016	1385	322	0	1814	294	3268
1047	833	232	1158	2147	0	0	3612	102	303
1656	1057	850	1756	1746	632	0	1752	3422	2466
0	0	835	0	0	2957	3612	0	1325	120
839	284	0	1081	1001	1065	2513	918	141	0
1696	2752	1068	2647	1832	1128	0	2077	819	1432
986	395	3275	1125	1312	123	319	1083	0	184

Tableau VI.55 - *Générateurs d'un code en blocs d.o. de taux* $\frac{8}{10}$

10	0
0	5
0	17
3	0
0	19
13	0
14	0
19	0

Tableau VI.56 – *Générateurs d'un code en blocs d.o. de taux* $\frac{8}{11}$

14	0	12
$\underline{0}$	$\underline{88}$	44
66	0	56
56	0	60
0	70	61
$\underline{88}$	$\underline{0}$	55
75	0	53
49	69	0

Tableau VI.57 – *Générateurs d'un code en blocs d.o. de taux* $\frac{8}{12}$

43	$\underline{239}$	27	$\underline{0}$
128	0	87	74
134	0	135	81
204	0	67	43
46	87	$\underline{0}$	$\underline{239}$
152	$\underline{0}$	$\underline{238}$	83
108	40	9	0
0	200	192	3

Tableau VI.58 – *Générateurs d'un code en blocs d.o. de taux* $\frac{8}{13}$

0	510	133	0	77
322	107	263	0	193
157	175	259	0	403
188	310	$\underline{0}$	$\underline{516}$	0
0	74	132	117	105
134	138	272	0	306
122	97	$\underline{515}$	$\underline{0}$	182
16	0	173	510	80

Tableau VI.59 - *Générateurs d'un code en blocs d.o. de taux* $\frac{8}{14}$

0	269	544	848	681	532
1053	0	269	0	558	268
0	152	625	1053	679	463
0	248	587	203	1047	496
403	303	317	0	358	433
0	256	941	210	541	558
0	709	526	450	521	528
514	338	0	1047	0	0

Tableau VI.60 - *Générateurs d'un code en blocs d.o. de taux* $\frac{8}{15}$

612	281	0	725	512	533	687
553	218	0	695	441	483	621
1530	846	0	792	936	465	597
216	0	737	1839	106	171	549
30	91	1575	116	1100	0	1576
1856	0	0	407	1083	338	576
0	816	1856	0	0	230	0
798	298	0	714	804	1261	702

Tableau VI.61 - *Générateurs d'un code en blocs d.o. de taux* $\frac{8}{16}$

151	58	65	2263	0	377	386	320
0	87	14	553	650	2179	207	505
2212	340	62	0	2660	0	0	0
0	2542	787	45	0	210	208	165
1104	82	104	185	0	537	497	2311
196	59	158	894	0	661	367	1413
88	0	0	124	0	1478	2659	288
1958	68	62	304	0	744	1900	925

Tableau VI.62 - *Générateurs d'un code en blocs d.o. de taux* $\frac{8}{17}$

0	4333	0	0	3916	2156	0	0	0
686	0	665	540	468	427	652	962	322
657	0	609	501	4333	374	2685	1099	1860
477	1289	401	378	3062	164	867	2036	0
675	0	1244	3797	1640	442	661	1940	265
4142	0	992	2369	426	532	1697	673	463
944	0	573	460	622	1342	737	1474	178
1678	1622	316	353	0	0	229	267	4333

Tableau VI.63 - *Générateurs d'un code en blocs d.o. de taux* $\frac{8}{18}$

0	5693	1082	4041	0	2344	1419	0	4639	0
1450	298	490	697	462	5157	0	549	1062	1040
529	5874	495	504	466	191	0	624	0	408
5484	0	0	0	3550	308	5874	1347	101	2674
582	471	507	535	1027	2037	0	1640	733	402
2113	489	596	1228	574	0	169	601	481	501
2896	265	631	493	772	0	10	490	286	2452
503	280	493	2055	494	671	0	5236	67	3565

Tableau VI.64 - *Générateurs d'un code en blocs d.o. de taux* $\frac{9}{11}$

0	12
28	0
0	27
0	8
26	0
0	3
14	0
0	9
0	19

Tableau VI.65 – *Générateurs d'un code en blocs d.o. de taux* $\frac{9}{12}$

0	50	74
0	101	63
1	124	0
122	0	12
0	41	57
0	27	36
0	7	18
74	0	40
3	0	123

Tableau VI.66 – *Générateurs d'un code en blocs d.o. de taux* $\frac{9}{13}$

7	160	0	42
16	69	0	50
335	0	42	0
45	0	203	3
342	0	0	24
18	126	0	82
151	46	0	55
0	41	45	270
0	33	343	129

Tableau VI.67 – *Générateurs d'un code en blocs d.o. de taux* $\frac{9}{14}$

0	166	218	247	335
0	743	117	43	19
0	151	145	212	702
784	0	0	54	0
0	744	353	298	174
0	265	784	551	185
0	196	159	224	179
769	70	539	0	333
109	0	68	57	2

Tableau VI.68 – *Générateurs d'un code en blocs d.o. de taux* $\frac{9}{15}$

0	78	1186	11	0	289
1280	365	368	589	981	0
223	81	117	162	87	0
352	342	1170	701	358	0
68	0	19	135	322	1153
9	0	24	33	132	1560
27	626	0	0	0	125
1341	394	231	230	256	0
12	1561	611	16	0	0

Tableau VI.69 – *Générateurs d'un code en blocs d.o. de taux* $\frac{9}{16}$

100	0	0	126	534	110	199
344	1972	451	565	0	277	1521
407	373	0	855	2633	402	457
0	11	2633	0	1395	0	0
369	2487	0	371	1385	1079	372
323	1291	0	382	168	703	2328
309	155	716	510	0	290	681
406	811	0	440	332	2455	413
292	156	24	2634	0	450	286

Tableau VI.70 – *Générateurs d'un code en blocs d.o. de taux* $\frac{9}{17}$

0	686	657	477	675	4142	944	1678
4333	0	0	1289	0	0	0	1622
0	665	609	401	1244	992	573	316
0	540	501	378	3797	2369	460	353
3916	468	4333	3062	1640	426	622	0
2156	427	374	164	442	532	1342	0
0	652	2685	867	661	1697	737	229
0	962	1099	2036	1940	673	1474	267
0	322	1860	0	265	463	178	4333

Tableau VI.71 - *Générateurs d'un code en blocs d.o. de taux* $\frac{9}{18}$

0	1082	380	5408	2908	226	589	966	671
0	350	187	3387	1434	559	581	607	5316
0	1606	50	482	203	830	447	471	23
3095	183	973	601	198	0	2896	479	23
1197	2840	92	438	163	0	419	1307	3027
0	5529	0	392	3887	230	379	950	0
0	1069	2256	609	361	145	1666	5707	304
0	454	128	580	5610	79	638	555	953
5707	0	65	0	0	1269	0	0	3338

Tableau VI.72 - *Générateurs d'un code en blocs d.o. de taux* $\frac{9}{19}$

61	64	168	315	8230	141	87	3417	103	0
22	2357	0	2059	73	2	39	8	1433	129
478	573	865	392	396	391	3338	1656	1446	0
0	5469	950	332	4097	7870	6385	45	7	1907
1	0	1262	75	0	8887	3934	0	0	9382
414	3282	605	414	434	2541	706	411	406	0
395	9381	1039	371	696	4617	548	1160	2545	0
0	2	0	132	6378	0	0	1723	549	4961
903	3556	9040	0	0	242	9	4273	9	1173

Tableau VI.73 - *Générateurs d'un code en blocs d.o. de taux* $\frac{10}{12}$

0	7
0	3
0	35
0	4
0	14
10	0
0	37
2	0
22	0
37	0

Tableau VI.74 – *Générateurs d'un code en blocs d.o. de taux* $\frac{10}{13}$

$\underline{0}$	83	$\underline{178}$
39	0	61
45	65	0
$\underline{178}$	18	$\underline{0}$
46	0	19
59	43	0
55	173	0
45	0	3
0	11	167
47	91	0

Tableau VI.75 – *Générateurs d'un code en blocs d.o. de taux* $\frac{10}{14}$

0	214	171	233
449	0	40	0
317	0	73	34
$\underline{467}$	36	$\underline{0}$	344
0	259	193	448
0	213	225	221
0	125	73	164
140	39	0	41
0	406	99	208
$\underline{0}$	350	$\underline{467}$	217

Tableau VI.76 – *Générateurs d'un code en blocs d.o. de taux* $\frac{10}{15}$

0	299	0	142	1135
0	320	146	107	794
0	679	5	977	172
87	333	0	106	310
0	418	60	123	99
0	945	210	90	82
1135	0	552	192	0
39	296	0	1003	97
387	198	193	0	0
132	208	1073	3	0

Tableau VI.77 – *Générateurs d'un code en blocs d.o. de taux* $\frac{10}{16}$

0	311	168	1916	453	520
0	775	0	268	270	286
780	2090	1445	437	30	0
1362	0	2089	668	244	253
0	206	222	527	440	822
0	81	90	610	341	346
85	79	0	224	745	2089
395	16	0	269	1783	1211
2023	2	168	0	0	8
776	19	0	414	574	299

Tableau VI.78 – *Générateurs d'un code en blocs d.o. de taux* $\frac{10}{17}$

1292	1047	1656	0	839	1696	986
723	833	1057	0	284	2752	395
484	232	850	835	0	1068	3275
3016	1158	1756	0	1081	2647	1125
1385	2147	1746	0	1001	1832	1312
322	0	632	2957	1065	1128	123
0	0	0	3612	2513	0	319
1814	3612	1752	0	918	2077	1083
294	102	3422	1325	141	819	0
3268	303	2466	120	0	1432	184

Tableau VI.79 – *Générateurs d'un code en blocs d.o. de taux* $\frac{10}{18}$

0	1450	529	5484	582	2113	2896	503
5693	298	5874	0	471	489	265	280
1082	490	495	0	507	596	631	493
4041	697	504	0	535	1228	493	2055
0	462	466	3550	1027	574	772	494
2344	5157	191	308	2037	0	0	671
1419	0	0	5874	0	169	10	0
0	549	624	1347	1640	601	490	5236
4639	1062	0	101	733	481	286	67
0	1040	408	26	4402	501	2452	3565

Tableau VI.80 - *Générateurs d'un code en blocs d.o. de taux* $\frac{10}{19}$

61	22	478	0	1	414	395	0	903
64	2357	573	5469	0	3282	9381	2	3556
168	0	865	950	1262	605	1039	0	9040
315	2059	392	332	75	414	371	132	0
8230	73	396	4097	0	434	696	6378	0
141	2	391	7870	8887	2541	4617	0	242
87	39	3338	6385	3934	706	548	0	9
3417	8	1656	45	0	411	1160	1723	4273
103	1433	1446	7	0	406	2545	549	9
0	129	0	1907	9382	0	0	4961	1173

Tableau VI.81 - *Générateurs d'un code en blocs d.o. de taux* $\frac{10}{20}$

1567	5917	658	3551	9830	174	0	3429	806	1787
10316	1317	656	1694	2128	1825	0	994	872	2492
3160	1305	1533	2878	1634	1689	0	962	1500	1685
1642	6999	820	2115	2927	1713	0	1276	854	4289
252	0	8826	373	660	419	1073	4121	0	648
8285	1320	5533	4892	3085	1908	0	2753	801	6252
1416	11352	0	9974	1009	1059	2550	373	170	2201
0	0	12426	0	0	0	0	5297	550	0
3424	798	0	4925	2839	1020	12426	291	119	11559
198	0	1195	5533	410	351	2725	0	6234	522